TurboDB Studio 4

Referenzhandbuch

dataweb.

Smart Database Technologies

TurboDB Studio

Die Turbo Datenbank

---

von Peter Pohmann, dataWeb

Inhaltsverzeichnis

1 Einsteigen

1.1 Die neue Version

1.1.1 Neu in TurboDB Studio

Gegenüber seinem Vorgänger Visual Data Publisher 3 bietet TurboDB Studio eine große Menge an Verbesserungen:

Erweiterungen in allen Versionen

- Völlig **neuentwickelte IDE** (Integrated Development Environment) mit einem einheitlichen Eigenschaftsfenster und Registern für alle Fenster
- Echte **Rollbalken** im Tabellenfenster
- Bis zu **254 Tabellen gleichzeitig** öffnen
- **Formatierte Memos** erlauben das Speichern von Texten in verschiedenen Schriften und Auszeichnungen
- **Interaktive Programmierhilfe** für die Makrosprache und Datenbankjobs: Eine Liste der verfügbaren Funktionen mit Erklärung und Parametern kann jederzeit eingeblendet werden.
- **Neuer Reportgenerator** mit vielen zusätzlichen Gestaltungsmöglichkeiten
- **Farbige Syntax-Hervorhebung** und Zeilennummern im Programm-Editor
- **Manipulieren der Formulare und Steuerelemente** durch Makros
- **Neue Steuerelemente** für die Formulargestaltung: Kalender-Control und mehr
- Spezielle **Formular-Module** zum direkten Ansprechen der Formulare und Steuerelement aus der Makrosprache heraus
- **ActiveX-Schnittstelle** zur Steuerung von **OLE-fähigen Anwendungen** und Bibliotheken wie z.B. Word oder Excel in Makros
- In Makros können die **Strings beliebige Länge** haben
- Ganze Datenbanken inklusive aller Tabellen und Indexe in einer **einzigen Datenbankdatei** ablegen
- Getestet mit **Windows XP** und **Windows Vista**
- Echter **Integer-Datentyp** sowie **Datum**, **Zeit** und **Zeitstempel**-Datentyp in der Makro-Programmierung
- Erhebliche **Geschwindigkeitssteigerungen** in der Datenbank-Engine
- Ausführen von **Makros geht erheblich schneller**
- **Neue Datentypen** in der Tabelle: **Integer**, **DateTime**, **Unicode**, **GUID**
- Arrays in der Makrosprache können bis zu **16 Megabyte** groß werden. Die neue Funktion Redim erlaubt ein **nachträgliches Ändern der Größe**.
- Die **Datei-Funktionen** einschließlich der Funktionen für den **RamText** können jetzt **beliebig lange Strings schreiben und lesen**. Die Beschränkung auf 255 Zeichen ist jetzt auch hier aufgehoben.
- Mit der **Systemvariablen Error** kann man im Programm den letzten Fehlercode und den dazugehörigen **Fehlertext** abfragen.
- Texte können jetzt **in allen Modulen und Datenbankjobs eines Projekts gesucht** werden.
- Module und Datenbankjobs werden jetzt in eine **einzige Programm-Datei** übersetzt.
- **Anker für Formularelemente** sorgen für eine flexiblere Ausrichtung beim Verändern der Formulargröße.
- **Typ-Konvertierung** in der Skript-Sprache erlaubt die kontrollierte Umwandlung von Variablenwerten

Zusätzliche Erweiterungen in der Professional Edition

- **Debugger** für Programme und Makros, beobachten Sie ihre Makros während der Ausführung Schritt für Schritt
- **Professionelle Installationsprogramme** mit dem frei verfügbaren Werkzeug WiX

Zusätzliche Erweiterungen in der Workgroup Edition

- Verwendung von **SQL-Befehlen** in Makros

Und hier die Änderungen gegenüber der Version VDP 3 im Detail:

Datenbank

- Möglichkeit alle Dateien (*.dat, *.mmo, *.blb, *.id, *.inr, *.in?, *.ind) in eine einzige Datenbankdatei zusammenzufassen
- Datentyp Integer für ganze Zahlen zwischen minus und plus zwei Milliarden
- Datentyp DatumZeit für kombinierte Angaben Datum und Uhrzeit
- Datentypen Unicode String und Unicode Memo für Texte in fremden, z.B. asiatischen oder arabischen Sprachen
- Über 200 Tabellen gleichzeitig öffnen statt wie bisher 62.
- Der Datenzugriff wurde gegenüber der Vorversion deutlich beschleunigt.
- In der Workgroup-Edition führt ein neuer Mechanismus dazu, dass keine blockierenden *.net- und *.mov-Dateien mehr übrig bleiben können.
- Alle Ganzzahltypen werden auch intern als Integer gespeichert, um Rundungsfehler zu vermeiden.

Formulare

- Tabellengitter haben jetzt einen echten Rollbalken statt der Rollknöpfe.
- Für die Eingabe von Datums-Werten steht jetzt ein Kalender-Control zur Verfügung.
- Formulare haben eigene Module, sogenannte Formular-Module, in denen ein direkter Bezug auf die Steuerelemente möglich ist.
- Alle Steuerelemente, können über Anker so eingerichtet werden, dass sie sich beim Vergrößern des Formulars korrekt mitbewegen.

Berichte

- Ein völlig neuer Reportgenerator verfügt über Dutzende neuer Funktionen.

Formulareditor

- Die Eigenschaften der einzelnen Elemente werden jetzt in einem einheitlichen Eigenschaftsfenster angezeigt, dass immer geöffnet bleibt.

Moduleditor

- Die Syntax-Hervorhebung im Moduleditor arbeitet jetzt mit verschiedenen Farben.
- Per Tastenkombination *[Strg]+[Leer]* kann man während des Programmierens eine Liste der verfügbaren Funktionen anzeigen lassen.
- Wenn man schon den Funktionsnamen eingetippt hat, erhält man per Tastenkombination *[Strg]+[Umschalt]+[Leer]* die nötigen Parameter für diese Funktion.
- Mit dem Quelltext-Debugger können Sie jetzt die Abarbeitung Ihrer Makros Schritt für Schritt beobachten, Haltepunkte setzen und Variableninhalte während der Laufzeit anzeigen.
- Der Befehl *Im Projekt* suchen ermöglicht es, einen Text in allen Modulen und Datenbankjobs eines Projektes zu suchen.
- Steht die Einfügemarke über einem TurboPL-Befehl, so wird mit *[F1]* die Hilfe zu diesem Befehl

geöffnet

Makrosprache

- Die Strings der Makrosprache können jetzt bis zu 2.000.000.000 Zeichen lang sein. Damit kann man ganze Memos durch Zuweisung in einen String speichern und auch wieder zurückschreiben. Umständliche Hilfskonstrukte wie Note und RamFiles sind dadurch nicht mehr nötig. (Aus Kompatibilitätsgründen gibt es sie aber noch.)
- Alle Module und Datenbankjobs eines Projektes werden jetzt in eine einzige Programm-Datei (*.prg) übersetzt, die ständig geladen bleibt. Somit geht das Aufrufen von Makros erheblich schneller als bisher. Außerdem bleibt der Wert von globalen Variablen bis zum Schließen des Projektes erhalten. Sie müssen bei der Auslieferung nicht mehr eine Liste von prg- und dbj-Dateien mitgeben sondern nur noch eine einzige prg-Datei.
- Die Funktionen Input, ChoseFile, ChoosePicture und ChooseFolder können jetzt auch mit beliebigen Variablen arbeiten und nicht nur mit T-Eingabe.
- Man kann nun einzelne Steuerelemente im Formular über die Makrosprache ansprechen und z.B. die Farbe setzen, aktivieren/deaktivieren oder sichtbar/unsichtbar machen.
- Die einzelnen Projektelemente sind jetzt über die Projekt-Variable ansprechbar.
- Mit dem neuen OleObject können OLE-fähige Anwendungen (Automatisierungs-Server) und ActiveX-Bibliotheken eingesetzt werden.
- In der Makrosprache steht nun ein echter Datentyp Integer für Variablen zur Verfügung, dadurch werden Rundungsfehler beim Rechnen mit ganzen Zahlen vermieden.
- Ebenfalls gibt es neue Datentypen für Datum, Uhrzeit und Zeitstempel. Dadurch werden Rundungsfehler beim Rechnen vermieden, der Debugger kann solche Werte lesbar anzeigen.

Sonstiges

- TurboDB Studio wurde auch mit Windows XP getestet.
- Das kostenlose Zusatztool TurboDB Workbench ermöglicht das Manipulieren von Datenbank-Tabellen auf der Kommandozeile.
- Ein weiteres kostenloses Zusatztool - TurboDB Viewer - ist ein einfacher Tabelle-Betrachter mit dem man aber auch Editieren kann, SQL-Abfragen ausführen sowie Tabellen erzeugen, restrukturieren und indizieren.
- Mit dem dritten kostenlosen Werkzeug, dataWeb Compound File Explorer, können Sie vorhandene Datenbank-Projekte mit getrennten Dateien für alle Tabellen auf den Modus für eine gemeinsame Datei umstellen (und auch wieder zurück).
- Mit TurboDB Studio wird auch eine Standard-Version von VDP 3 ausgeliefert. Falls Sie keine Lizenz von VDP 3 besitzen, benötigen Sie dieses Programm, um Ihre Projekte auf den Stand der Version 3 zu bringen. Nur dann kann es von TurboDB Studio geöffnet werden.
- Für TurboDB ist jetzt auch ein ODBC-Treiber verfügbar.
- Mit gesondert erhältlichen Komponenten-Sammlungen können Sie den Zugriff auf TurboDB-Tabellen auch in Ihre Delphi, Kylix, Visual Basic.NET und C#-Applikationen einfach integrieren.

Nicht mehr enthalten

Die folgenden Komponenten sprengen nach heutigem Stand den Rahmen einer Desktop-Datenbank und sind deshalb nicht mehr enthalten:

- Erzeugen von Web-Applikationen. Dafür hat dataWeb eine eigene Entwicklungsumgebung herausgebracht, den dataWeb Builder.
- Erstellen einer Schablone für eine Hilfe-Datei. Für solche Aufgaben gibt es eine ganze Reihe von spezialisierten Werkzeugen, die auch Ihr vorhandenes Hilfe-Projekt einlesen können. Wir empfehlen hier KAL von Linn edv und Help & Manual von EC Software.
- Installations-Programm: TurboDB Studio erstellt jetzt Projekte für das frei verfügbare Werkzeug WiX. Damit können Sie professionelle MSI Installationsprogramme mit weit mehr Optionen als bisher gestalten.

1.1.2 Neue und erweiterte Skript-Befehle

Variablentypen

Integer	Ganze Zahlen von -2^63 bis +2^63. Vermeidet Rundungsfehler beim Rechnen und Vergleichen.
Date	Datum
Time	Uhrzeit mit einer Genauigkeit bis Mikrosekunden ohne Rundungsfehler
DateTime	Zeitstempel

Kontrollstrukturen

Neben *wiederhole/repeat* und *while/solange*-Schleifen sind nun auch for-Schleifen möglich. Diese erhöhen automatisch den Schleifenzähler bis zu einem maximalen Wert.

Typ-Konvertierung

Mit dem neuen Schlüsselwort *as* werden Variablen eines Typs in Variablen eines anderen Typs konvertiert.

Funktionen

BitXor	Kombiniert zwei Integer-Werte per exklusivem oder
BitNot	Berechnet die Invertierung eines Bit-Wertes
BitShl	Verschiebt die Bits eines Integers nach links
BitShr	Verschiebt die Bits eines Integers nach rechts
NewGuid	Erzeugt einen neuen weltweit eindeutigen Bezeichner für eine TurboDB Guid-Spalte.
FirstRec	Liest den ersten Datensatz einer Tabelle
PrevRec	Liest den vorhergehenden Datensatz.
NextRec	Liest den nächsten Datensatz.
LastRec	Liest den letzten Datensatz.
PostRec	Schreibt einen geänderten Datensatz in die Datenbank.
InArray	Funktioniert jetzt mit Integer, Real und String-Feldern
DateTimeStr	Berechnet einen String aus einer Zeitstempel-Angabe
DateTimeVal	Berechnet einen Zeitstempel aus einem String
LocalToUTC	Berechnet die UTC Zeit aus der lokalen Zeit.
UTCToLocal	Berechnet die lokale Zeit aus der UTC Zeit.
MakeTime	Berechnet einen Datums/Zeit-Wert aus Stunde, Minute, Sekunde und Millisekunde.
MakeDate	Berechnet einen Datums/Zeit-Wert aus Jahr, Monat und Tag.
MakeDateTime	Berechnet einen Datums/Zeit-Wert aus den Einzelangaben.
TimeStr	Hat einen zusätzlichen optionalen Parameter für die Genauigkeit
OpenSQL	Öffnet eine SQL Ergebnismenge.
ExecSQL	Führt einen SQL-Befehl aus.
Trace	Schreibt einen Text in das Hinweisfenster der Entwicklungsumgebung.
DeleteTable	Löscht eine Datenbanktabelle
CopyBlob	Schreibt den Blob-Inhalt in eine Datei
ClearBlob	Löscht ein Blob.
DelIndex	Löscht einen Index.
FindTable	Ermittelt die Tabellen-Nummer zu einem Tabellennamen.
EnumStr	Liefert den Text eines Aufzählungswertes zu seiner Nummer.
EnumVal	Berechnet Nummer eines Aufzählungswertes zu seinem Namen.
GetType	Enthält jetzt zusätzliche Angaben zum Datentyp

Redim	Erlaubt das Ändern der Dimensionen eines Arrays

Klasse DataWnd

CurrentRecNo	Liefert die Nummer des im Datenfensters angezeigten Datensatzes.
RowNum	Liefert die Anzahl der im Datenfenster angezeigten Datensätze.

Klasse Control

Diese Klasse ist neu und umfasst alle Steuerelemente auf einem Formular.

1.1.3 Upgrade auf TurboDB Studio

TurboDB Studio ist eine stark verbesserte und in vielen Bereichen gründlich überarbeitete Version von TurboDB Studio 3. Deshalb bleibt es nicht aus, dass einige Detail aus Ihrem VDP 3-Projekt geändert werden müssen, um mit TurboDB Studio fehlerfrei zu arbeiten. Die folgende Liste stellt eine kurze Übersicht über die notwendigen Änderungen dar, eine ausführliche Anleitung finden Sie im Abschnitt "VDP-Anwendungen für TurboDB Studio anpassen".

- TurboDB Studio kann nur die **Projekt-Dateien von VDP 3** lesen. Projekte von **VDP 2 und davor werden nicht mehr akzeptiert**. Falls Sie ihr Projekt mit einer älteren Version erstellt haben, benutzen Sie den im Paket enthaltenen VDP 3 Standard, um das Projekt auf den aktuellen Stand zu bringen. Dazu öffnen Sie einfach das Projekt in VDP 3. Dann starten Sie den Formular-Editor für jedes Formular im Projekt und wählen den Menüpunkt Speichern. Nun können Sie das Projekt in VDP 3 wieder schließen und in TurboDB Studio öffnen.
- VDP 3 und alle Vorgänger setzen auf **Datentabellen das ReadOnly-Flag**, wenn diese geschlossen werden. Die Idee dabei war, dass ein versehentliches Löschen nicht so einfach geschehen kann. Dieses Verfahren ist aus mehreren Gründen nicht mehr zeitgemäß. Zum einen zeigt Windows heute beim Löschen einer Datei selbst eine Sicherheitsabfrage an. Zweitens gibt es vielfältige Möglichkeiten des Zugriffsschutzes, mit denen Datensicherheit erreicht werden kann. Andererseits verhindert das ReadOnly-Flag auf einem Web-Server oft, dass man die Datei editieren oder löschen kann. Somit haben wir uns entschlossen, ab TurboDB Studio das ReadOnly-Flag genau zu dem zu verwenden, wozu es auch gedacht ist. **Tabellen mit ReadOnly-Flag können zwar betrachtet aber nicht geändert werden**. Wenn Sie ein altes Projekt mit TurboDB Studio öffnen, werden Sie deshalb gefragt, ob etwaige ReadOnly-Flags auf den Datenbank-Dateien entfernt werden sollen.
- Bisher konnte man **Memo-Feldern Zahlen zuweisen**, z.B:

  ```
  MEINETABELLE.Bemerkung := 0;
  ```

  Dies ist aber eine gefährliche Programmiertechnik. Mit obigem Aufruf entsteht z.B. ein "verwaistes" Memo in der Memo-Datei, welches außer bei Restrukturierung nicht mehr freigegeben werden kann. Deshalb sind solche Zuweisungen nicht mehr erlaubt. Erlaubt ist:

  ```
  MEINETABELLE.Bemerkung := "";
  ```

- TurboDB Studio unterstützt nun erheblich besser als VDP 3 **globale Variablen in Modulen**. Alle Module des Projektes bleiben vom Übersetzen an komplett im Speicher, wodurch die globalen Variablen ihren Wert behalten, bis das Projekt geschlossen oder neu übersetzt wird. Eine Folge davon ist, dass globale Variablen in mehreren Modulen nicht den selben Namen haben dürfen. Sollten Sie globale Variablen mit dem gleichen Namen in mehreren Modulen des selben Projektes benutzen, **müssen Sie sie entsprechend umbenennen**. Beim Übersetzen weist Sie TurboDB Studio auf solche Variablen hin.
- Die **Objektorientiert Programmierung** wurde in TurboDB Studio erweitert und ausgebaut. Eine Folge davon ist, dass bestimmte eigentlich falsche Konstrukte, die in VDP 3 noch akzeptiert wurden nun nicht mehr übersetzt werden. Z.B:

  ```
  vardef obj: Object;
  obj := FindDataWnd('XXX');
  obj.CloseWnd;
  ```

  Da obj "nur" ein ganz normales Objekt ist und kein Datenfenster, kann es auch kein CloseWnd ausführen. In VDP 3 wurde diese Sequenz allerdings noch ausgeführt, in TurboDB Studio müssen Sie es richtig schreiben:

  ```
  vardef obj: DataWnd;
  ```

```
obj := FindDataWnd('XXX');
obj.CloseWnd;
```

- Die Funktion ***ExecMacro*** ist durch das neue Programm-Konzept, bei dem alle Module immer geladen sind **überflüssig geworden**. Sie sollten Aufrufe wie

  ```
  ExecMacro(MeinModul, DieProzedur(8))
  ```

  ersetzen durch

  ```
  MeinModul.DieProzedur(8)
  ```

Diese Aufrufe sind nicht nur wesentlich weniger fehleranfällig sondern vor allem auch erheblich schneller, weil dabei nicht erneut ein Programm in den Speicher geladen werden muss. Die Funktion ExecMacro funktioniert allerdings weiterhin in den meisten Anwendungsfällen. Sie müssen nur darauf achten, dass das aufgerufene Modul im Projekt enthalten ist. ExecMacro-Aufrufe in Modulen werden teilweise nicht mehr übersetzt und sollten durch direkte Aufrufe ersetzt werden. (Siehe auch ExecMacro.)

- Wenn Sie Funktionen aus DLLs aufrufen, die Strings erwarten, müssen Sie entweder in der Deklaration ein *as LPStr* ergänzen oder nach aktuellem Windows-Standard die **Version für Unicode-Strings** benutzen. Bei den Standard-Windows-Funktionen ist das die Version mit W am Ende statt wie bisher mit A.
- **Doppelte Prozedurnamen innerhalb eines Moduls** sowie solche, die einen Punkt im Namen enthalten, wurden bisher nicht als Fehler betrachtet. Ab TurboDB Studio schon. Außerdem war es bisher teilweise möglich, Prozeduren aus anderen Modulen zu verwenden, ohne diese mit Uses oder Include einzubinden. Jetzt erhalten Sie hier eine Fehlermeldung und müssen ein Uses ergänzen.
- Wenn die Funktionen *Exists*, *LinkSum* und *LinkCount* auf die Primärtabelle angewendet wurden, haben sie keine Schleife über die Tabelle durchgeführt, sondern nur den aktuellen Datensatz berücksichtigt. Diese wurde erweitert, so dass man jetzt mit ***Exists*, *LinkSum* und *LinkCount* auch über die Primärtabelle suchen**, summieren und zählen kann. Für *Link*, *LinkSum* und *LinkCount* gibt es neue alternative Namen: *LoopRecs*, *SumRecs*, *CountRecs*.
- Ausdrücke in Formelfeldern und Zeilen in Modulen und Datenbankjobs konnten schon bisher **nicht mehr als 255 Zeichen umfassen**. In VDP 3 und früher wurden solche Ausdrücke stillschweigend abgeschnitten. Jetzt wird eine Fehlermeldung angezeigt.
- Die Funktion *ReadLn* hat bisher keinen Fehler gemeldet, wenn der **Dateizeiger schon am Ende der Datei** angelegt war. Jetzt kommt in diesem Fall wie auch bei *Read* ein Lesefehler.
- In früheren Versionen konnte man **hinter eine Zeile beliebige Kommentare** schreiben, z.B.

  ```
  ReadRec(A, 1); Den ersten Datensatz lesen.
  ```

  Hier muss jetzt wie auch sonst ein Kommentarzeichen stehen, also:

  ```
  ReadRec(A, 1); ..Den ersten Datensatz lesen.
  ```

  Dies gilt auch für vermeintliche Variableninitialisierung:

  ```
  vardef a: Integer = 2;
  ```

  Hier wurde das = 2 in VDP 3 als Kommentar betrachtet und deshalb nicht ausgeführt. TurboDB Studio meldet jetzt einen Fehler, damit die Situation klar ist.
- Die **Abkürzungtaste Strg+F9 zum Ausführen eines Moduls** oder Datenbankjobs wurde der einfacheren Bedienung wegen durch F9 ersetzt. Strg+F9 dient jetzt zum Übersetzen des ganzen Projekts.
- Die folgende Befehlssequenz hat in VDP 3 dazu geführt, dass der **neue Datensatz im Datenfenster editiert** wurde:

  ```
  ReadRec(0)
  WriteRec(FileSize+1)
  EditRec
  ```

  Das ist aber eigentlich nicht korrekt und dementsprechen, wird in TurboDB Studio jetzt der Datensatz editiert, der vor Aufruf dieser Sequenz im Datenfenster selektiert war. Wie sonst auch, braucht man auch in diesem Fall jetzt ein Attach, um den neuen Datensatz zu editieren:

  ```
  ReadRec(0)
  WriteRec(FileSize+1)
  Attach
  EditRec
  ```

- Der **Datentyp *TBits*** wurde aus Konsistenzgründen in *Bit* umbenannt.
- Frühere Versionen von TurboDB ließe **doppelte Spaltennamen in Tabellen** zu, d.h. es war möglich zwei Spalten der selben Tabelle z.B. *Name* zu nennen. Das dies natürlich immer wieder zu Problemen führte (welches ist gemeint wenn auf Feld *Name* verwiesen wird?), prüft TurboDB Studio dies beim Öffnen einer Tabelle ab und zeigt ggf. eine Meldung an. Sie sollten im Sinne einer größeren Klarheit diese Felder umbenennen, z.B. in Name1 und Name2.
- Da es die neue Systemvariable *Project* bzw. *Projekt* gibt, werden **Spaltennamen einer Tabelle namens PROJEKT** nicht mehr als solche erkannt. PROJEKT.Name führt zur Fehlermeldung "unbekannter Bezeichner: Name". Mit einem $ davor geht es wieder: $PROJEKT.Name.
- Die Funktionen ***DateStr* und *DateTimeStr*** liefern jetzt für Werte kleiner als 367 die Jahreszahl 0: DateStr(1) -> 1.1.0000
- Die Funktion ***Choice/xWert* benötigt eigentlich mindestens zwei Argumente**. Falls nur ein Argument angegeben war, lieferte sie bisher einen zufälligen Wert zurück. Jetzt kommt beim Übersetzen ein Fehler.
- Der **Steuerbefehl HL** hat das Argument in alten Versionen fälschlicherweise nicht in der mit MM eingestellten Maßeinheit interpretiert und deshalb war die Länge der Linie abhängig von der Auflösung des Druckers. Jetzt hat die **Linienlänge die selbe Maßeinheit wie alle anderen Angaben** im Datenbankjob.
- Die Syntax von Eingabemustern hat sich geändert.
- Die Funktion **GetStars** erwartet als Parameter jetzt ein Integer-Array, statt bisher ein Real-Array.

1.2 Schnelleinstieg

1.2.1 Was ist TurboDB Studio?

TurboDB Studio ist ein Desktop Datenbankprogramm vergleichbar etwa mit **Microsoft Access (R)** oder **FileMaker (R)**. Mit TurboDB Studio können Sie ohne weitere Zusatz-Tools **Datentabellen** aufbauen und verknüpfen, Daten editieren und auswerten sowie **Formulare, Berichte und Datenjobs** für Ihre Daten gestalten. Zudem ist TurboDB Studio eine **Programmierumgebung**, mit der Sie anspruchsvolle Makros und ganze Applikationen entwickeln und weitergeben können. Dazu enthält TurboDB Studio eine **Makrosprache**, **Programmeditor** mit integriertem **Quelltext-Debugger** und die Möglichkeit, jederzeit aus dem Projekt eine **ausführbare Datei** zu erstellen.

TurboDB Studio ist der Nachfolger von **Visual Data Publisher 3**. Wir haben den Produktnamen und einige andere Bezeichnungen innerhalb des Programms geändert, weil mit dieser Version die Betonung wieder in erster Linie auf Datenbank-Entwickler gelegt wird und der Publishing-Aspekt etwas in den Hintergrund gerät. Die Bezeichnung TurboDB Studio soll einerseits die Verwandtschaft mit unseren anderen TurboDB-Produkten wie **TurboDB für Delphi, TurboDB.NET und TurboDB ODBC** betonen. Andererseits steht der Begriff Studio seit einiger Zeit als Synonym für eine Entwicklungsumgebung, siehe Visual Studio, Delphi Studio usw.

Schließlich möchten wir TurboDB Studio nun auch außerhalb des deutschsprachigen Raums vertreiben und müssen deshalb die englischen Bezeichnungen mit mehr Vorsicht verwenden. Als prominentesten Ausdruck in dieser Reihe haben wir die Makrosprache easy in **TurboPL** (Turbo Programming Language) umbenannt.

1.2.2 Was ist das Besondere an TurboDB Studio?

Bei der Entwicklung von TurboDB Studio (ehemals Visual Data Publisher) haben wir uns besonders darüber Gedanken gemacht, wie man die doch recht anspruchsvolle Aufgabe der Datenbankentwicklung so gestalten kann, dass nicht mehr viele Wochen nötig sind, um eine Anwendung zu erstellen. An diesem Ziel arbeiten wir seit mehreren Jahren und haben eine ganze Reihe von Lösungen gefunden, die TurboDB Studio von anderen Datenbank-Werkzeugen abheben:

Datensätze suchen

Wenn Sie mit TurboDB Studio arbeiten, brauchen Sie sich nicht mit SQL oder ähnlich komplizierten Abfragesprachen herumärgern. Stattdessen formulieren Sie Suchbedingungen wie etwa: *Preis größer 80* oder *Termin ist 18.10.2000* oder *Name wie "Müll*"*. Das meiste davon können Sie aus vorhandenen Listen auswählen.

Datensatz-Ids

In vielen Datenbanken geistert immer noch das Thema Primärschlüssel zur eindeutigen Identifizierung von Datensätzen herum. Die in TurboDB Studio integrierte Turbo-Datenbank verfügte als eine der ersten Datenbank überhaupt über die automatische Vergabe von Datensatz-Ids, die das Problem der Primärschlüssel ein für allemal erledigt.

Tabellen verknüpfen

Eines der schwierigsten Themen im Datenbankbereich ist die korrekte Verknüpfung von Informationen aus mehreren Tabellen. Meistens muss man bei jeder Abfrage wieder neu formulieren, wie die verschiedenen Tabellen zusammenhängen. Tatsache ist aber, dass diese Zusammenhänge fast immer die gleichen sind. Z.B. werden Rechnungen nun mal über die Kundenummer dem Empfänger zugeordnet und ein Buch hat eine Verknüpfung zu seinen Autoren. Deshalb werden in der Turbo-Datenbank die meisten Verknüfpungen zwischen Tabellen schon beim Anlegen der Tabelle definiert. Das spart viel Aufwand bei zukünftigen Auswertungen und bringt weitere Vorteile: So kann z.B. schon bei der Eingabe von Datensätzen automatisch überwacht werden, dass verknüpfte Datensätze auch wirklich existieren.

Markierungen

Meistens arbeitet man in einer Datenbank ja nicht mit allen Daten, sondern mit einer Auswahl, die mich gerade interessiert. Diese Auswahl muss aber voll bearbeitbar sein, z.B. alle unbezahlten Rechnungen, alle ausgeliehenen Bücher, alle Kunden mit einer bestimmten Vorliebe usw. Turbo-Datenbank arbeitet hier mit einem sehr flexiblen Konzept von zweistufigen Markierungen. Damit gibt es ständig vier verschiedene Datenbereiche: Alle Datensätze, die angezeigten Datensätze, die markierten Datensätze und der aktuelle Datensatz. Diese vier Ebenen sind eine großartige Hilfe bei der Arbeit mit Untermengen von Tabellen.

Volltext-Suche

Neben der herkömmlichen Suche mit Suchbedingung bietet Turbo-Datenbank auch die Volltextsuche nach einem beliebigen Stichwort an. Dazu können pro Tabelle beliebig viele Volltext-Indexe erstellt werden, die sich auf eine unterschiedliche Auswahl von Spalten der Tabelle beziehen. Dadurch ist die Volltext-Suche genauso schnell wie die herkömmliche Suche.

Netzwerk-Betrieb

Wenn mehrere Anwender die selben Daten in einem Netzwerk gleichzeitig benutzen, kann das verschieden Schwierigkeiten hervorrufen. Turbo-Datenbank sorgt hier mit ausgeklügelten Mechanismen dafür, dass alles automatisch reibungslos vor sich geht. So hat z.B. jeder Anwender immer den Überblick, wer gerade noch die selben Daten bearbeitet. In Konfliktfällen, wird dem Benutzer eine genau Beschreibung der Situation gegeben und die freie Wahl gelassen, ob er z.B. auf seine Änderungen verzichten oder auf die Freigabe des Datensatzes durch den anderen Bearbeiter warten möchte.

User-Modus

Datenbank sind zu komplex als dass ein Endanwender direkt damit arbeiten könnte. Praktisch jede Datenbank braucht eine angepasste Oberfläche mit Vorgaben und Prüfungen, damit der Benutzer sie sinnvoll einsetzen kann. Diese Oberfläche erzeugen Sie mit TurboDB Studio im Handumdrehen. Das Geheimnis liegt darin, dass Sie nicht bei Null anfangen sondern, dass Ihre Anwendung von Beginn an eine automatisch generierte umfangreiche Oberfläche besitzt. Zu einem großen Teil, besteht Ihre Aufgabe also nur im Weglassen von nicht gewünschten Funktionen. Zusätzlich können Sie über einen einfachen Editor die Menüs und Schalterleisten Ihrer Anwendung bearbeiten und eigene Funktionen hinzufügen. Das ganze wird mit einem einzigen Befehl in eine EXE-Datei übersetzt und kann so benutzt werden. Auch eine Vorlage zur Erzeugung eines Installationsprogramms wird automatisch erstellt.

1.2.3 Der erste Start

Angenommen, Sie sitzen zum ersten Mal vor *TurboDB Studio*, haben ein paar Menüpunkte angeklickt und möchten nun wissen, wie Sie am schnellsten und einfachsten lernen, was das Programm kann und wie man es macht. Wenn Sie ein systematischer Typ sind und etwas Zeit haben, empfehlen wir Ihnen, das Tutorial. Das gibt Ihnen in einer logischen Abfolge von Schritten eine Anleitung zum praktischen Arbeiten mit *TurboDB Studio*.

Wenn Sie gleich "richtig" anfangen wollen, raten wir Ihnen dazu, ein Projekt aus dem Unterverzeichnis *Beispiel* Ihres *TurboDB Studio*-Installations-Verzeichnisses zu öffnen. Am einfachsten geht das im Willkommen-Fenster, das beim Start von TurboDB Studio angezeigt wird. Wählen Sie hier *Ein Beispielprojekt öffnen* und suchen Sie sich aus der Liste eines aus. Oder Sie klicken im Menü *Datei/Beispielprojekt öffnen...* an und suchen das gewünschte Projekt selbst im Ordner Beispiel.

Projekte

Wenn Sie das Projekt geöffnet haben, sehen Sie eine Liste von Projektelementen in dem Projektfenster auf der rechten Seite. Dabei handelt es sich um Tabellen und die zugehörigen Formulare, Berichte, Datenbankjobs und Module. Sie können diese Elemente an ihren Symbolen unterscheiden und außerdem in der Statuszeile die Elementart ablesen.

Mit jedem dieser Elemente sind - z.B. über die Schalter im Projektfenster - zwei prinzipielle Aktionen möglich: Ausführen (der grüne Pfeil) und Entwerfen (Lineal und Stift). Wenn Sie ein Formular oder eine Tabelle ausführen, können Sie die Daten der zugehörigen Tabelle ansehen und editieren. Berichte und Datenbankjobs produzieren beim Ausführen eine Ausgabe auf dem Drucker oder in eine Datei (siehe Kapitel "Ausdrucke produzieren"). Beginnen Sie am besten damit, ein Formular auszuführen. Es wird geöffnet und zeigt den ersten Datensatz der Tabelle. Mit den Pfeilschaltern der Schalterleiste können Sie zwischen den Datensätzen wechseln. Wenn Sie Daten ändern möchten, müssen entweder *Datensätze bearbeiten* oder *Neue Datensätze eingeben* im Menü *Bearbeiten* wählen. Um die Daten (oder einen Ausschnitt davon) auszudrucken, verwenden Sie einen *Bericht* oder einen *Datenbankjob*. Ein Doppelklick im Projektfenster führt ihn aus. Sie haben die Wahl, ob das Ergebnis in der Vorschau, auf dem Drucker oder in eine Datei ausgegeben werden soll.

Entwerfen von Elementen bedeutet, dass Sie die Form des Formulars, die Gestalt des Ausdrucks oder die Struktur der Tabelle verändern können. Probieren Sie es einmal mit dem Formular. Der Befehl *Entwerfen* im lokalen Menü des Projektfensters öffnet den Formulareditor. Hier können Sie die Eigenschaften der einzelnen Felder verändern, die Felder verschieben oder löschen und neue Felder einfügen. Zum Einfügen suchen Sie in der Komponenten-Palette zuerst das richtige Register heraus und ziehen dann die gewünschte Komponente in das Formular. Wenn Sie eine Tabelle entwerfen, bestimmen Sie die Struktur der Daten, d.h. Name, Typ und Anzahl der Tabellenspalten. Im Tabelleneditor können Sie neue Spalten einfügen und vorhandene löschen. Gelöschte Spalten sind unwiederbringlich verloren, weil Sie hier nicht nur die Darstellung am Bildschirm, sondern tatsächlich die Daten löschen. Mehr Information dazu finden Sie in den entsprechenden Abschnitten wie z.B. Formulare und Berichte gestalten.

Situationsgerechte Hilfe

Damit Sie mit TurboDB Studio möglichst bald effizient arbeiten können, werden Ihnen in allen denkbaren Situationen Informationen angeboten.

- Die Statuszeile enthält in verschiedenen Abschnitten immer einen kurzen Hinweis zu Ihrer aktuellen Situation. Dies kann eine Hilfemeldung des gerade selektierten Menüpunktes, der Zustand der Groß- oder Überschreibtaste oder auch nur die Anzahl der Datensätze in der gerade geöffneten Tabelle sein.
- Die Schalter der Schalterleiste verfügen über ein gelbes Hinweisfenster, das erscheint, wenn man mit der Maus längere Zeit darauf verweilt.
- Wenn Sie die [F1]-Taste drücken, erhalten Sie eine ausführliche Erklärung zu Ihrer aktuellen Situation in der Windows-Hilfe. Hier ist jeder Menüpunkt, jeder Schalter, jede Meldung und jeder Dialog erläutert.
- Im Moduleditor erhalten Sie mit [Strg]+[F1] eine Erklärung zum Befehl an der aktuellen Kursorposition.
- Auch bei weitergehenden Fragen hilft die Online-Hilfe. Wenn Sie einen Lösungsweg für eine

bestimmte Aufgabe suchen, sehen Sie im entsprechenden Abschnitt, z.B. "Formulare gestalten" nach. Falls Ihnen ein Kommando oder eine Funktion der Skriptsprache entfallen ist, schlagen Sie bitte in der TurboPL Referenz nach. Und schließlich finden Sie im Glossar alle wichtigen Begriffe kurz erklärt.

- Als nächstes können Sie auch im Internet Unterstützung finden. Auf der Web-Seite von dataWeb (www.dataweb.de) finden Sie Informationen und ein Diskussionsforum zu TurboDB Studio.
- Als nächsten Schritt können Sie ein E-Mail an unseren Support schreiben: support@dataweb.de
- Falls alle Stricke reißen, helfen wir Ihnen natürlich auch in Form von Projekt-Unterstützung und Programmierung gegen Honorar weiter.

1.2.4 Häufig gestellte Fragen

Oberfläche

Warum reagiert TurboDB Studio nicht auf Eingaben im Formular oder der Tabelle?

Um Datensätze zu ändern, müssen Sie in den Editiermodus wechseln (*Bearbeiten/Datensätze ändern*). Zur Eingabe neuer Datensätze dient der Neueingabemodus (*Bearbeiten/Neue Datensätze* eingeben).

Warum finde ich einen Menüpunkt nicht mehr, obwohl ich ihn gerade noch gesehen habe?

Die Menüs des *TurboDB Studio* sind kontextsensitiv. D.h. je nach aktuellem Fenster haben Sie andere Befehle zur Auswahl. Sie müssen also zuerst in das richtige Fenster klicken, dadurch passt sich die Menüzeile und die Schalterleiste an. Häufig passiert das beim Projektfenster, weil es immer neben allen anderen Fenstern angezeigt wird. Die Befehle des Projektfensters sind aber ebenso erst dann verfügbar, wenn Sie es angeklickt haben.

Wie kann ich in einem mehrzeiligen Eingabefeld im Formular einen Absatz eingeben?

Mit [Strg]+[Enter]. Die [Enter]-Taste alleine genügt nicht, weil man damit von einem Feld zum nächsten gelangt.

Wie kann ich am einfachsten die Datensätze ausdrucken, die ich im Datenfenster markiert habe?

Dazu gibt es zwei Möglichkeiten. Bei der ersten Möglichkeit muss der Bericht als Sortierung *Markierte Datensätze* eingestellt haben, bzw. der Datenbankjob im Prolog das Kommando *.ACCESS Markierung* enthalten. Dann genügt es, wenn Sie im Formular oder im Tabellenfenster die gewünschten Datensätze markieren und dann unter Ausführen den Bericht oder Datenbankjob starten.

Bei der zweiten Möglichkeit enthält der Datenbankjob kein *ACCESS*-Kommando und der Bericht ist auf *Vorhandene Sortierung verwenden* eingestellt. Dann müssen Sie im Datenfenster die Anzeige auf *Nur markierte Datensätze* einstellen, bevor Sie wiederum unter Ausführen Ihren Datenbankjob oder Bericht starten.

Wie bekomme ich meine Tabellen aus der Turbo-Datenbank für DOS in ein TurboDB Studio-Projekt?

Wählen Sie *Datei/Projektelement hinzufügen...* im Hauptmenü, selektieren Sie dann den Eintrag *Tabelle* als Dateityp im Dialogfeld und suchen Sie die gewünschte Tabelle.

Programmierung

Wie kann ich feststellen, ob eine bestimmte Tabelle geöffnet ist?

Verwenden Sie die Funktion *FindTable*. Wenn das Ergebnis größer als Null ist, ist die Tabelle schon geöffnet.

Wie kann ich ermitteln, wieviele Datensätze bei einer Suche gefunden wurden?

Das hängt natürlich von der Art der Suche ab. Wenn Sie mit Datenbank-Befehlen wie *Link/LoopRecs* suchen und die passenden Datensätze markieren, können Sie anschließend mit *NMarks* deren Anzahl bestimmen. Wenn Sie in der Oberfläche mit *MitBedingung/BySelection* suchen, dann verwenden Sie entweder *StarNum* oder *RowNum*. *StarNum* liefert die Anzahl der Oberflächenmarkierungen und wird deshalb eingesetzt, wenn nach der Suche nicht auf die markierten Datensätze umgeschaltet wurde. *RowNum* liefert die Anzahl der angezeigten

Datensätze und darum die richtige Wahl, wenn nach der Suche auf die markierten Datensätze umgeschaltet wurde.

1.2.5 Lizenzierung

Es gibt mehrere Arten, wie Sie ihre TurboDB Studio Vollversion erhalten können.

Über das Internet

Dies ist für Sie und für uns die schnellste und günstigste Alternative:

1. Laden Sie sich die aktuelle Version von TurboDB Studio von der Web-Site von dataWeb herunter: http://www.turbodb.de/de/ordering/downloads.html
2. Bestellen Sie eine Lizenz. Dies können Sie ebenfalls auf unserer Web-Site tun, der Kauf wird dann über den renommierten Dienstleister Share-It abgewickelt, bei dem Sie per Kreditkarte, Vorauszahlung und bei Firmenkunden auch per Rechnung bezahlen können. Sie erhalten innerhalb von wenigen Stunden eine Lizenzdatei per E-Mail zugesandt. Sie können hier auch angeben, dass Sie eine CD zugesandt haben möchten.
3. Schalten Sie die heruntergeladene Version frei: Im Windows-Startmenü finden Sie unter TurboDB Studio auch den Befehl *TurboDB 4 Studio Aktivieren*. Starten Sie den Aktivator und geben Sie den Pfad zur Lizenzdatei an. Der Aktivator schaltet die installierte Version mit den Daten aus der Lizenzdatei frei.

Wenn Sie ein Update von TurboDB Studio aus dem Internet holen, wiederholen Sie einfach den dritten Schritt.

Per Post

Sie können aber auch auf herkömmlichem Weg bestellen:

1. Senden Sie eine Bestellung per Internet oder per Post direkt an dataWeb. Wir akzeptieren Vorauskasse, Nachnahme, Bankeinzug und bei uns bekannten Kunden auch Rechnung. Die Internet-Bestellung erreichen Sie über *http://www.turbodb.de/de/ordering/direct.html*. Für die Post-Bestellung öffnen Sie im Startmenü unter TurboDB Studio das Bestellformular, füllen es aus und senden es an die angegebene Adresse.
2. Sie erhalten per Post entweder eine CD mit allen Angaben oder einen Brief mit dem Lizenzschlüssel und Installationsanleitung.

Auch in diesem Fall können Sie gegebenenfalls eine Update-Version mit den vorhandenen Lizenzdaten und dem mitgelieferten Aktivator jederzeit freischalten.

Windows Vista

Auf Grund der erhöhten Sicherheitsvorkehrungen unter Windows VISTA, muss der Aktivator im Administrator-Modus gestartet werden. Dazu müssen Sie als System-Administrator angemeldet sein. Starten Sie den Aktivator dann nicht wie gewohnt durch einfachen Mausklick im Start-Menü, sondern öffnen Sie mit der rechten Maustaste das Kontextmenü und wählen Sie den Befehl *Als Administrator ausführen*.

1.2.6 Support & Adressen

Auf den Web-Seiten der dataWeb GmbH finden Sie ein Forum für alle Fragen rund um TurboDB Studio und andere TurboDB Produkte. Dieses Forum wird von dataWeb-Mitarbeitern betreut, die keine Anfrage ohne Antwort lassen. Das sollte Ihre erste Anlaufstelle sein, wenn Sie Fragen, Anregungen oder Probleme haben:

http://www.dataweb.de/de/support/

Auch auf der Web-Präsenz der TDB Software Service GmbH in Schwabach finden Sie ein gut besuchtes Forum zum Thema Visual Data Publisher (der Vorgängerversion von TurboDB Studio):

http://www.plaudern.de/a.prg?nap=1

Darüberhinaus leistet dataWeb einen kostenfreien E-Mail Support unter der E-Mail-Adresse:

support@dataweb.de

Falls Sie Schulungen im Bereich *TurboDB Studio* und *TurboDB für VCL* oder *.NET* besuchen

möchten bzw. Beratungs- oder Entwicklungs-Leistungen benötigen, wenden Sie sich bitte direkt an die dataWeb GmbH: (Unter dieser Telefonnummer und Postanschrift können wir keinen Programm-Support leisten.)

dataWeb GmbH

Hofmarkstr. 40

94529 Aicha

Tel: +49 85 44 97 18 90

Fax: +49 85 44 97 18 99

Email: info@dataweb.de

Internet: http://www.dataweb.de

2 Tutorial

2.1 Ein Streifzug durch TurboDB Studio

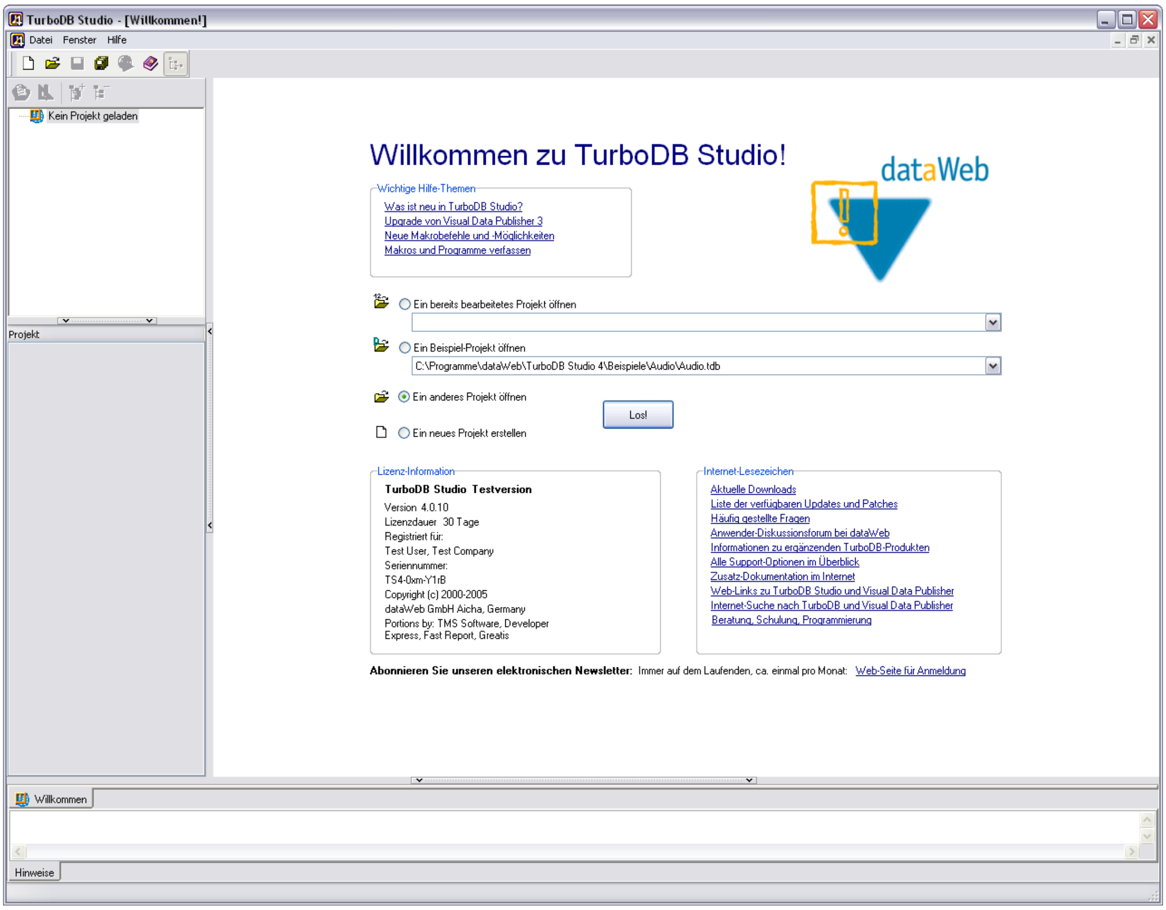
In diesem Kapitel möchten wir Sie auf einen Streifzug durch *TurboDB Studio* einladen. Von den allerersten Anfängen bis zu anspruchsvollen Aufgaben und dem Erstellen eigener Applikationen führen wir Sie Schritt für Schritt durch das Programm.

Als Beispiel betrachten wir die Anwendung *KFZ-Tutorial*, die Sie im Installationsverzeichnis von *TurboDB Studio* im Unterverzeichnis *Beispiele/Tutorial* finden.

2.2 Schritt 1: Ein Projekt öffnen

Wenn Sie *TurboDB Studio* gestartet haben, sehen Sie zunächst das *"TurboDB Studio"-Logo*, dann das *Rahmenfenster*, welches Sie mit der *Willkommens-Seite* begrüßt und nach Ihren Wünschen fragt. Mit ihrer Hilfe können Sie auf einfache Art und Weise entscheiden, ob Sie mit der Bearbeitung des zuletzt geöffneten Projektes fortfahren wollen, ein anderes Projekt geöffnet werden soll, Sie mit einem Beispielprojekt experimentieren oder aber ein völlig neues Projekt erstellen wollen. Außerdem finden Sie hier noch nützliche Internet-Lesezeichen zum Thema *TurboDB Studio*.

Wenn Sie die erste Option wählen, so wird das im Textfeld angegebene Projekt geladen. Für den Fall, dass Sie ein neues Projekt beginnen wollen, unterstützt Sie ein Assistent beim Erstellen einer neuen Datenbankanwendung. Doch dazu später mehr.



Das TurboDB Studio Rahmenfenster zeigt nach dem Start die Willkommens-Seite

Wir wollen uns zuerst einer bereits bestehenden Anwendung widmen: Dem *KFZ-Tutorial*-Projekt, das mit *TurboDB Studio* als Beispiel-Anwendung ausgeliefert wird. Markieren Sie also den Punkt *Ein Beispiel-Projekt öffnen* und wählen Sie aus der Liste das Projekt *(...)Beispiele/Tutorial/KFZ-Tutorial.tdb* aus. Der Gesamtpfad in diesem Listeneintrag hängt von Ihrem Rechner und den Angaben bei der Installation ab. Mit einem Klick auf *Los!* wird das Beispiel-Projekt geöffnet.

Als nächstes sitzen Sie vor dem Projektfenster von *KFZ-Tutorial*.

2.3 Schritt 2: Infos und Hilfe

Nachdem das Projekt geladen wurde, sehen Sie das *Rahmenfenster* mit dem darin enthaltenen *Projektnavigator*. Bevor wir uns diesem zuwenden, soll in diesem Abschnitt aber zunächst erläutert werden, wie Sie sich in verschiedenen Situationen bei Fragen online Antworten verschaffen können.

Riskieren wir einmal einen Blick auf das Menü. Es ist kontextsensitiv, d.h. es ändert sich in Abhängigkeit von dem gerade aktiven Fenster und bietet immer die für dieses Fenster passenden Befehle an. Darunter befindet sich die Schalterleiste mit Knöpfen, die einen schnellen Zugriff auf die wichtigsten Funktionen ermöglichen. Beachten Sie, dass einzelne Schalter auch deaktiviert sein können, je nachdem, ob ihre Funktion gerade Sinn macht oder nicht. Was die einzelnen Schalter bedeuten, können Sie ganz leicht herausfinden, indem Sie mit dem Mauszeiger darauf zeigen. In der *Statusleiste* erscheint dann eine kurze Erklärung. Wenn Sie noch etwas länger warten, kommt die Erklärung auch noch in einem kleinen gelben Fenster unterhalb des Mauszeigers.

Windows-Hilfe

Um Hilfe zur aktuellen Situation zu erhalten gibt es eine allgemeine und einfache Regel. Ein Druck

auf die *[F1]*-Taste öffnet die Windows-Hilfe. Wenn die Einfügemarke auf einem bekannten Befehl steht, dann wird die Hilfe für diesen Befehl aufgerufen. Doch dazu später mehr.

Eine andere Art, in die Windows-Hilfe zu gelangen, bildet das Hilfe-Menü. Hier können Sie entweder das *Inhaltsverzeichnis* oder eines der angeführten *Hilfethemen* direkt anwählen.

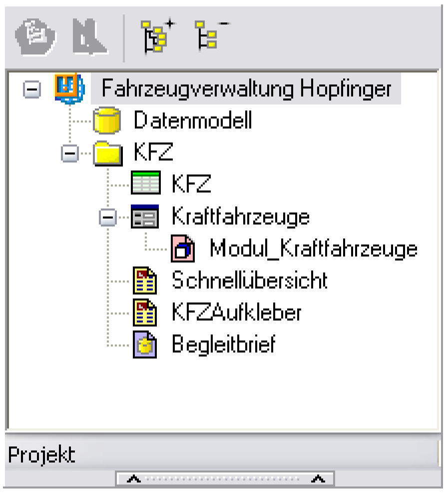
Wie man die Windows-Hilfe bedient, wissen Sie bereits wenn Sie schon einmal mit Windows-Anwendungen gearbeitet haben. Mit den Pfeiltasten oder dem Mausrad bzw. Rollbalken können Sie sich durch das gerade angezeigte Thema bewegen. Mit den Pfeilschaltern am oberen Fensterrand des Hilfe-Fensters gelangen Sie zum jeweils vorherigen oder nächsten Hilfe-Thema. Besonders wichtig sind die Wörter und Ausdrücke, die im Text hervorgehoben sind. Sie bezeichnen Bezüge auf andere Themen der Hilfe, zu denen Sie durch einen Klick mit der linken Maustaste gelangen. Alternativ dazu können Sie die Referenz mit der *Tabulatortaste* markieren und dann die *[Enter]*-Taste drücken.

Von diesen Referenzen gibt es zwei Arten: Die einen sind unterstrichen dargestellt und führen Sie direkt zum zugehörigen Thema. Die anderen sind punktiert unterstrichen und zeigen das referenzierte Thema in einem zusätzlichen Fenster nur solange an, bis Sie eine Taste drücken.

In der Online-Hilfe möchten wir Ihr Augenmerk besonders auf den Teil *"Aufgaben"* richten, den Sie direkt über die Inhalt-Seite auswählen können. Hier finden Sie Schritt-für-Schritt-Anleitungen zu vielen Problemstellungen der täglichen Arbeit.

2.4 Schritt 3: Projekte

Grundlegend für die Arbeit mit *TurboDB Studio* sind die *Projekte*, in denen alle Dateien für eine bestimmte Aufgabe zusammengefaßt werden. Hier verwalten Sie die zusammengehörenden Tabellen mit ihren Formularen und sonstigen Elementen. Auf dem Bildschirm präsentiert sich das Projekt in seiner Gesamtheit im *Projektnavigator*.



Das Beispiel-Projekt besteht aus der Tabellen KFZ (grünes Symbol), dem Formular Kraftfahrzeuge (graues Symbol) mit dem zugehörigen Modul (hellrotes Symbol), zwei Berichten (gelbe Symbole) und einem Datenbankjob (lila Symbol).

Wenden wir uns dem Aufbau des Beispiel-Projekts *KFZ-Tutorial* zu: In der obersten Zeile sehen Sie das Projektsymbol mit dem *Titel des Projekts*, dem alle anderen Einträge untergeordnet sind. Der erste Eintrag des Projekts ist das *Datenmodell*, das in jedem Projekt vorhanden ist. Darunter sehen Sie ein Ordnersymbol namens *KFZ* und darunter mehrere Titel mit verschiedenen Symbolen. Der Ordner trägt immer den Namen der Datentabelle und enthält alle zu dieser Tabelle gehörenden Dateien. Das oberste Symbol im Ordner steht für die *Datentabelle KFZ* die die eigentlichen Informationen speichert. In diesem Beispiel sind es die Beschreibungen verschiedener Fahrzeugtypen mit Bezeichnung, Hersteller, Baujahr usw. Darunter schließen sich mehrere Elemente an, die zur Bearbeitung dieser Daten benötigt werden. Da ist in der nächsten Zeile das *Formular Kraftfahrzeuge*, mit dem Sie die Daten betrachten und bearbeiten können. Dem Formular untergeordnet sehen Sie das *Formularmodul Modul_Kraftfahrzeuge*, das kurze *TurboPL*-Programme (Makros genannt) enthalten kann. Dann kommt der *Bericht Schnellübersicht*, der eine einfache tabellarische Auflistung aller gespeicherten Autos produziert. Anschließend

sehen Sie den *Bericht Aufkleber*, zum Drucken von Etiketten mit den Fahrzeugdaten, und schließlich ist da noch der *Datenbankjob* Begleitbrief, der eine einseitige Beschreibung für jedes Fahrzeug auf dem Drucker ausgibt. Da alle diese Elemente sich direkt auf die in der Datentabelle erfassten Autos beziehen, liegen Sie im *Ordner KFZ*.

Wenn Sie mit der Maus oder den Pfeiltasten einen Eintrag im *Projektnavigator* selektieren, sehen Sie in der Statuszeile des *Projektnavigators* den Typ des selektierten Elements.

Ein Projekt kann natürlich auch jeweils mehrere Elemente vom selben Typ beinhalten, z.B. zwei Berichte für zwei verschiedene Etiketten-Formate. Und selbstverständlich besitzt ein Projekt meistens auch mehrere Tabellen. Später in diesem Streifzug werden Sie noch eine Tabelle der Käufer und eine für die Bestellungen hinzufügen.

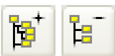
Über der Liste mit den Projektelementen befinden sich vier Schalter mit den wichtigsten Befehlen. Die Schalter-Aktion bezieht sich immer auf das gerade selektierte Element.



Öffnen / Ausführen startet das Projektelement. Tabellen und Formulare werden geöffnet, damit Sie Daten betrachten und bearbeiten können. Berichte und Datenbankjobs werden beim Ausführen ausgegeben. Je nach Wahl entweder auf den Drucker oder in die Druckvorschau bzw. eine Datei.



Mit *Entwerfen* können Sie Aufbau, Aussehen und Eigenschaften des Projektelements verändern. Dazu öffnet sich bei Tabellen der TurboDB Tabellen Designer, bei Formularen der Formulareditor, bei Berichten der Berichteditor und für Module und Datenbankjobs ein Texteditor.



Mit diesen beiden Symbolen können Sie *alle Projektelemente aufklappen* bzw. *alle Projektelemente zuklappen*. Das ist nützlich, wenn Sie sich einen Überblick über das Projekt verschaffen wollen.

Wenn Sie ein Projektelement mit der rechten Maustaste anklicken, dann erscheint ein Kontextmenü, das neben den vier oben erklärten Symbolen noch fünf weitere enthält.



Der Menüpunkt *Neu* dient dazu, das Projekt um neue Elemente zu ergänzen. Sie können ein neues Projektelement erstellen und in das aktuelle Projekt aufnehmen.



Der Schalter *Hinzufügen* dient dazu, das Projekt um vorhandene Elemente zu ergänzen. Sie können Projektelemente, die Sie für andere Projekte erstellt haben, in das aktuelle Projekt mit aufnehmen.



Mit *Entfernen* können Sie Dateien aus dem Projekt ausschließen ohne die Datei von der Festplatte zu löschen.



Datei löschen entfernt die markierte Datei aus dem Projekt und löscht sie von der Festplatte.



Der Menüpunkt *Datenbank-Explorer anzeigen* zeigt ein Fenster in dem die Struktur der Datentabellen detailliert angezeigt werden können.

2.5 Schritt 4: Das Tabellenfenster

Zunächst aber sollten Sie sich die bestehende Tabelle einmal näher ansehen. Selektieren Sie diese im Projektfenster und drücken Sie den Knopf *Ausführen*. Das lokale Menü (Kontextmenü) öffnen Sie mit der rechten Maustaste, solange sich die Maus über dem Projektfenster befindet. Die schnellste Variante jedoch stellt ein Doppelklick mit der linken Maustaste auf die selektierte Tabelle dar. Für welche Methode Sie sich auch entscheiden: in jedem Falle öffnet sich nun das Tabellenfenster.

KFZ

Bezeichnung	Modelljahr	Hersteller	Art	Motor	Leistung	Kilowatt	Hubraum	Höchstgeschwindigkeit	Servo	ABS	Katalysator	Sitze	Rabatt-Gewährung	Brutto-Preis	Rabatt-Satz
Alpha Spider	1978	Alpha L...		B..	101	74	1948	193	☐	☐	☐	4	☑	8900,00	15,00
Audi 80 qu...	1995	Audi AG	K..	Be...	170	125	2300	215	☑	☑	☑	5	☐	45670,00	8,50
BMW 318 i	1984	Bayeris...	L...	Be...	105	77	1766	184	☐	☐	☐	5	☐	50000,00	4,50
BMW 320 i	1989	Bayeris...	L...	Be...	129	95	1990	195	☐	☐	☑	5	☑	56300,00	12,00
BMW M 5	1989	Bayeris...	L...	Be...	316	232	3535	250	☑	☑	☑	5	☐	28500,00	0,00
Ford Capri	1982	Ford-W...	C..	Be...	90	66	1998	173	☐	☐	☐	5	☐	25000,00	0,00
Mercedes ...	1971	Daimler...	C..	Be...	150	111	2500	185	☑	☐	☐	8	☑	56549,50	1,25
Ford Sierra L	1984	Ford-W...	L...	Be...	76	56	1593	165	☐	☐	☐	8	☐	32000,00	0,00
Lada 1200	1981	VAZ (L...	L...	Be...	60	44	1189	140	☐	☐	☐	8	☐	12250,00	0,00
Mercedes ...	1981	Daimler...	L...	Die...	60	44	1988	135	☐	☐	☐	5	☐	42300,00	0,00
Mercedes ...	1984	Daimler...	K..	Be...	136	101	2299	180	☑	☐	☐	8	☐	62500,00	0,00
Mercedes ...	1981	Daimler...	L...	Be...	140	104	2525	185	☑	☐	☐	8	☐	54500,00	0,00
Mercedes ...	1989	Daimler...	L...	Be...	160	118	2597	200	☑	☑	☑	8	☐	59900,00	0,00
Mercedes ...	1989	Daimler...	L...	Die...	103	76	2996	190	☑	☑	☐	8	☐	72000,00	0,00
Opel Asco...	1981	Adam ...	L...	Be...	60	44	1297	145	☐	☐	☐	8	☑	22800,00	15,00
Opel Kadett	1981	Adam ...	L...	Be...	53	39	1169	140	☐	☐	☐	8	☐	19900,00	0,00
Opel Ome...	1989	Adam ...	L...	Be...	115	85	1998	195	☑	☐	☑	8	☐	35280,00	0,00
Porsche 9...	1989	F. Pors...	C..	Be...	250	185	3600	260	☑	☑	☑	4	☑	156000,00	10,00
VW Golf C...	1981	Volksw...	C..	Be...	95	70	1781	166	☐	☐	☑	5	☐	34054,00	0,00
VW Golf GL	1989	Volksw...	L...	Be...	55	41	1272	151	☐	☐	☑	8	☐	38000,00	0,00
VW Golf G...	1982	Volksw...	C..	Be...	110	81	1588	182	☐	☐	☐	8	☑	34000,00	12,00
VW Passa...	1963	Volksw...	K..	Die...	54	40	1588	143	☐	☐	☐	8	☑	42000,00	5,70
VW Passa...	1989	Volksw...	K..	Be...	72	53	1595	172	☐	☐	☑	5	☐	28350,95	0,00
VW Polo F...	1989	Volksw...	K..	Be...	45	33	1043	142	☐	☐	☐	8	☑	21830,00	2,75
VW Polo LS	1981	Volksw...	K..	Be...	50	37	1093	145	☐	☐	☐	8	☐	18000,00	0,00
Suzuki Sw...	1986	Suzuki ...	L...	Be...	250	185	1500	0	☐	☐	☐	8	☐	14850,00	0,00
Alpha Spider	1981	Alfa-La...	L...	Be...	126	93	1948	193	☐	☐	☐	5	☑	8900,00	10,00
Audi 80 qu...	1993	Audi AG	K..	Be...	170	125	2300	215	☑	☑	☑	5	☐	45670,00	8,50
Ford Capri	1952	Ford-W...	C..	Be...	90	66	1998	173	☐	☐	☐	5	☐	25000,00	0,00
VW Passa...	1982	Volksw...	K..	Die...	54	40	1588	143	☐	☐	☐	8	☑	41000,00	25,00
Jaguar 345/T	1900	United ...	C..	Be...	90	66	2000	130	☐	☐	☐	4	☑	120000,00	0,50
Ford Capri	1981	Ford-W...	C..	Be...	90	66	1998	173	☐	☐	☐	5	☐	24500,00	0,00

Im Tabellenfenster sehen Sie den Inhalt Ihrer Datentabelle.

Die oberste Zeile enthält die Namen der Spalten (auch Datenfelder genannt) und dann folgen die Beschreibungen der Fahrzeuge, immer eine pro Zeile. In der Tabellensicht ist immer genau ein Datensatz selektiert. Sie erkennen Ihn an dem gestrichelten Rahmen um die Zeile und dem nach rechts deutenden Pfeil in der Markierungsleiste auf der linken Seite. Den Pfeil können Sie mit den *Pfeiltasten* oder der *[Bild auf]* und *[Bild ab]*-Taste verschieben und so jeden Datensatz auswählen. Auch die Pfeil-Schalter in der Schalter-Leiste und der *Rollbalken* am rechten Fensterrand dienen zu diesem Zweck. Natürlich reicht auch ein Mausklick auf die entsprechende Zeile. Mit den mit  und  gekennzeichneten Schaltern gelangen Sie zum ersten und zum letzten Datensatz der Tabelle.

Eine Spalte der selektierten Zeile enthält die Einfügemarke. Diese können Sie frei innerhalb der Tabelle bewegen. Mit der *[Tab]*-Taste rückt man die Einfügemarke eine Spalte nach rechts und mit der Kombination *[Umschalt]+[Tab]* eine Position nach links. Befindet sie sich vor einer solchen Aktion bereits am Anfang bzw. Ende einer Zeile, so gelangt man damit in die letzte Spalte bzw. erste Spalte.

In der Statuszeile am unteren Rand des Rahmenfensters sehen Sie fünf Felder:

Das kleinste befindet sich ganz links und enthält einen Stern, wenn der aktuelle Datensatz markiert ist. Dies erreichen Sie durch *[Strg]+Klick* auf den Datensatz, die Taste *[F8]* oder den Menüpunkt *Suchen/Markierung umschalten*. Das zweite Feld der Statuszeile zeigt die Nummer des Datensatzes bezogen auf die angezeigten Datensätze an. Diese Nummer ist abhängig von der Datenselektion und der Sortierung der angezeigten Datensätze. Im dritten Feld wird die Nummer des aktuellen Datensatzes bezogen auf die Reihenfolge in der Tabelle angezeigt. Diese Nummer ist unabhängig von der Anzahl der angezeigten Datensätze und ihrer Sortierung. Das vierte Feld zeigt, wieviele Datensätze aus der Tabelle angezeigt werden können. Diese Zahl ändert sich entsprechend einer eingestellten Datenselektion, was im fünften Feld der Statuszeile durch die Angabe *Alle* bzw. *Auswahl* verdeutlicht wird.

	Sequenznummer 1	Satznummer 1	Datensätze 35	Alle

Die Statuszeile des Tabellenfensters zeigt verschiedene Informationen zum aktuellen Datensatz und der angezeigten

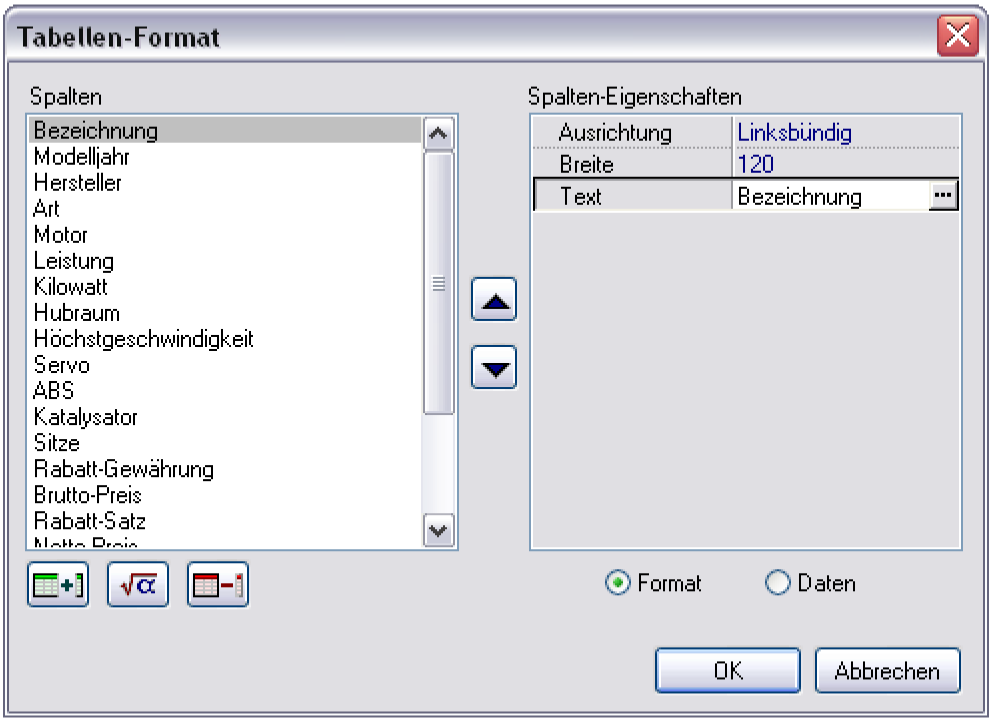
Datenmenge

Weil die Tabelle breiter als das Fenster ist, müssen Sie sie mit dem unteren Rollbalken hin- und herschieben, damit Sie alle Einträge sehen können. Auch mit der schon erwähnten Tabulatortaste können Sie durch den gesamten Tabelleninhalt rollen.

Der anfängliche Aufbau der Tabellensicht entspricht genau der Struktur der Datentabelle. Die Überschriften lauten genauso wie die Bezeichnungen der Datenfelder und die Spalten stimmen auch in Reihenfolge und Breite mit den Tabellenfeldern überein. Wenn Ihnen die Breitenverhältnisse der Spalten nicht zusagen, können Sie die Trennlinien mit der Maus verschieben. Zeigen Sie mit dem Mauszeiger auf eine solche Linie zwischen den Spaltenbeschreibungen der Tabelle. Der Mauszeiger verwandelt sich in das Symbol zum Verschieben der Linie. Drücken Sie nun die linke Maustaste und ziehen Sie die Linie an die gewünschte Stelle. Alternativ können Sie auch einen Doppelklick auf die Trennlinie zwischen den Spalten ausführen, dann wird die Breite automatisch am längsten Feldinhalt ausgerichtet.

Auf eine ähnliche Art können Sie übrigens auch die Reihenfolge der Spalten ändern: Sie brauchen dazu nur die Spalte an die gewünschte Position zu ziehen. Dabei wird die Reihenfolge Daten der zugehörigen Tabellenspalten natürlich nicht geändert. Manipulationen an der Tabellensicht ändern immer nur die Darstellung.

Zum Ändern einer Überschrift oder dem Inhalt einer Spalte finden Sie im Menü *Ansicht* den Menüpunkt *Format*. Damit aktivieren Sie das Dialogfenster für das Tabellenformat, in dem Sie alle Angaben einstellen können. In der linken Liste finden Sie alle angezeigten Spalten. Die aktuelle Spalte ist selektiert. Auf der rechten Seite finden Sie eine Liste aller Einstellungen, die Sie ändern können.



Im Tabellen-Format Dialog können Sie die Darstellung der Daten im Tabellenfenster ändern

Im Dialogfeld für das Tabellenformat können Sie auch Spalten entfernen und hinzufügen. Mit dem Schalter  wird eine Liste aller in der Datentabelle vorhandenen Spalten angezeigt. Markierten Sie die Spalten die hinzugefügt werden sollen (Mehrfachauswahl mit *[Strg]+Klick*). Mit dem Schalter  wird die in der Liste markierte Spalte entfernt und somit nicht mehr angezeigt. Mit dem Schalter  können Sie auch Formeln zur Berechnung von Spalten eingeben. Fügen Sie dazu mit dem Schalter eine Spalte hinzu, und wechseln Sie dann bei Optionsfeld unter der rechten

Liste auf die Option *Daten*. Hier können Sie die Berechnungsvorschrift bei *Formel* eintragen.

Sämtliche Änderungen können mit dem Menüpunkt *Ansicht/Standard-Format* wieder auf die Vorgabewerte zurückgesetzt werden.

Lassen Sie uns die Überschrift der Spalte *Leistung* in PS ändern. Klicken Sie dazu das Tabellenfenster an und wählen Sie anschließend den Menüpunkt *Ansicht/Format...*. Klicken Sie im Dialogfenster in der linken Liste die Spalte Leistung an und tragen Sie im Feld *Text* auf der rechten Seite *PS* ein. Nun könnten Sie noch die Spaltenbreite eingeben und bei Spalten mit Fließkommazahlen die Anzahl der Dezimalstellen angeben. Bestätigen Sie mit *Ok* und Ihre Änderungen werden durchgeführt.

Lassen Sie uns jetzt einen neuen Datensatz an die Tabelle anhängen. Dazu schalten Sie mit *Bearbeiten/Neue Datensätze eingeben* in den Neueingabe-Modus um. Ein Blick auf die Schalterleiste zeigt, dass der Schalter für die Neueingabe (eine stilisierte Tabelle mit 'Anhängsel') nun ebenfalls arretiert ist. Sie hätten auch mit diesem Schalter die Neueingabe aktivieren können.

Am Ende der Tabelle wird dabei eine neue Zeile mit leeren Spalten angefügt. Dort können Sie nun Ihre neuen Informationen eintragen.

Klicken Sie zum Ausprobieren einmal in die Spalte *Bezeichnung* und geben Sie eine Fahrzeugbezeichnung ein, z.B. Audi A6. Bestätigen Sie die Eingabe mit der *[Enter]*-Taste. Daraufhin wechselt der Kursor zur Spalte *Modelljahr*. Hier können Sie das gewünschte Jahr eintippen, z.B. 2005. Wenn Ihre Eingabe keine Zahl darstellt, erhalten Sie einen Hinweis.

Tragen Sie nun noch den *Hersteller* Audi AG ein und bestätigen Sie auch dieses wieder mit *[Enter]*. Wenn Sie in der Spalte *Art* etwas eintippen, so klappt eine Liste nach unten. Das liegt daran, dass hier nur ein Wert aus dieser Liste eingetragen werden kann. Diese *Werteliste* sehen Sie auch, wenn Sie mit der Maus auf den Knopf  klicken. Wählen Sie nun eine der möglichen Werte aus und bestätigen Sie. Die Spalten *Leistung*, *Kilowatt*, *Hubraum* und *Höchstgeschwindigkeit* erwarten nun wieder Zahlenwerte als Eingabe.

Bei den Spalten *Servo*, *Abs* und *Katalysator* taucht ein Markierungsfeld für Ja/Nein-Werte auf. Hier kann man nur zwischen  für Ja oder  für Nein wählen.

In den verbleibenden Spalten begegnet uns nichts Neues mehr. Aber sicherlich fällt Ihnen auf, dass *Laufende_Nummer* bereits einen Wert enthält. Diesen können Sie auch nicht verändern, da er von *TurboDB Studio* für interne Zwecke benötigt wird, wie Sie in einem der späteren Kapitel noch sehen werden.

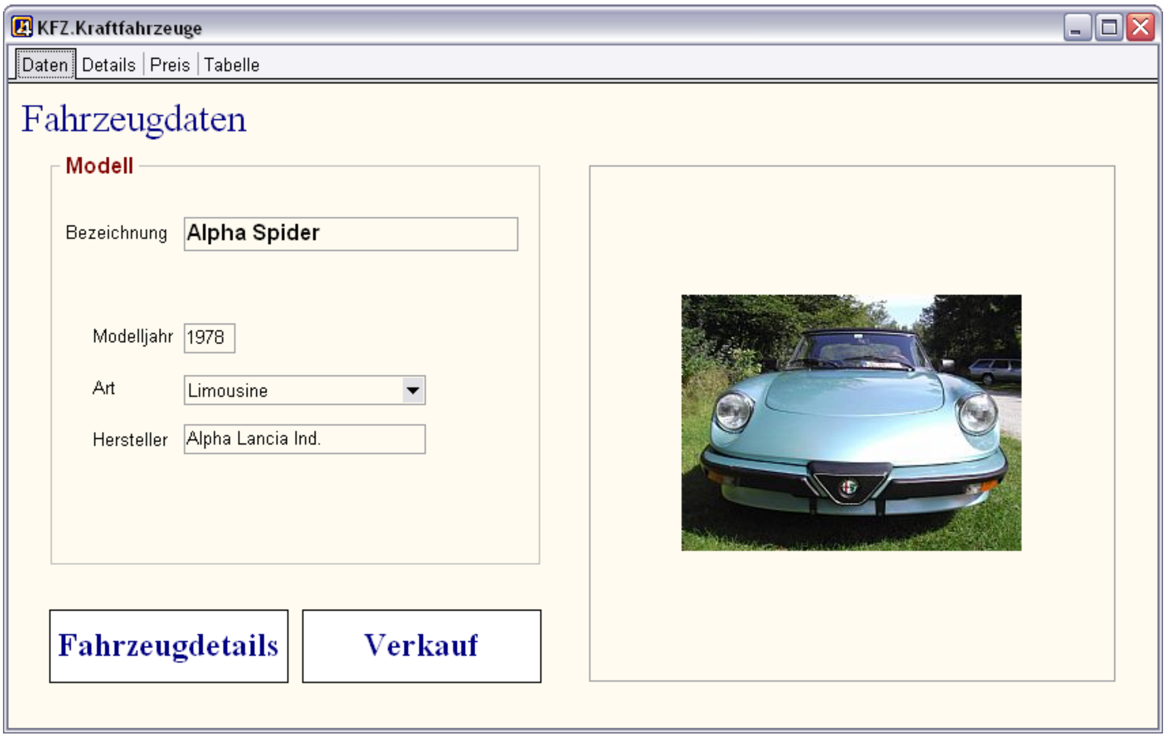
Um die Neueingabe zu beenden, lösen Sie am besten den Knopf in der Schalterleiste aus seiner Arretierung.

Sollten Sie nun bemerken, dass Sie sich in irgendeiner Spalte vertippt haben, so besteht die Möglichkeit den betreffenden Datensatz zu ändern. Dazu schalten Sie den Editiermodus ein, indem Sie *Bearbeiten/Datensätze ändern* wählen, die Abkürzungstasten *[Strg]+[E]* drücken oder den entsprechenden Schalter der Schalterleiste anklicken. Das weitere Vorgehen ist identisch mit dem bei der Dateneingabe. Wenn Sie beim Editieren bemerken, dass Sie sich geirrt haben, können Sie mit *Bearbeiten/Rückgängig* den ursprünglichen Zustand des Satzes wiederherstellen. Der Editiermodus wird verlassen, wenn Sie den entsprechenden Schalter aus der Arretierung lösen.

2.6 Schritt 5: Formulare

Eine für den Bearbeiter wesentlich angenehmere Darstellungsform von Datensätzen bieten Formulare. Selektieren Sie jetzt das Formular *Kraftfahrzeuge* und öffnen Sie es auf die gleiche Weise wie das Tabellenfenster.

Es erscheint das Formular mit der Darstellung des ersten Datensatzes. Wenn das Formular mit einer eingebetteten Tabelle versehen ist, können sie mit dem Menüpunkt Ansicht/Formularsicht bzw. mit dem Formularansicht-Schalter oder der Taste [F7] zwischen der ersten Seite des Formulars und der Seite mit der Tabelle wechseln. Enthält das Formular keine Tabelle, so sind der Menüpunkt und der Knopf in der Schalterleiste deaktiviert.



Ein Formular zeigt den aktuellen Datensatz optisch ansprechend aufbereitet und kann zusätzliche Eingabehilfen enthalten

Wenn Sie sich im Formular auf der ersten Seite befinden, ist der Schalter für die Formularsicht in der Schalterleiste arretiert, um den Sichtwechsel zu verdeutlichen. Wenn Sie ihn aus seiner Arretierung lösen, würden Sie zur im Formular auf dem *Registerblatt Tabelle* vorhandenen Tabellensicht wechseln.

Im Formular ist also meist nur ein Datensatz zu sehen. Dafür ermöglichen Rahmen, Beschriftungen, Logos und Schalter jedoch eine wesentlich strukturiertere Darstellungsmöglichkeit. In unserem speziellen Fall haben Sie sogar nicht einmal den ganzen Datensatz vor sich, sondern nur die erste Seite eines mehrseitigen Formulars. Zum Wechseln zwischen den Seiten stehen Ihnen neben den Tasten *[Alt]+[Bild auf]* und *[Alt]+[Bild ab]* sowie den Schaltern der Schalterleiste vor allem die Registerreiter am oberen Rand des Formulars zur Verfügung. Mit *[Tab]* und *[Umschalt]+[Tab]* gelangen Sie von einem Datenfeld zum nächsten. Zwischen den Datensätzen bewegen Sie sich wieder mit den Pfeilschaltern oder der Tastenkombination *[Strg]+[Bild ab]* für den nächsten und *[Strg]+[Bild auf]* für den vorherigen Datensatz.

Als nächstes werden wir einen neuen Datensatz an die Tabelle anhängen. Wie im Tabellenfenster schalten Sie mit *Bearbeiten/Neue Datensätze eingeben* in den Neueingabe-Modus um. Ein Blick auf die Schalterleiste zeigt, dass der Schalter für die Neueingabe (eine stilisierte Tabelle mit 'Anhängsel') nun ebenfalls arretiert ist. Sie hätten auch mit diesem Schalter die Neueingabe aktivieren können.

Der neue Datensatz ist leer und dementsprechend zeigt das Formular keine Feldinhalte. Das erste Datenfeld *Bezeichnung* hat nun einen weißen Hintergrund um anzuzeigen, dass Sie hier etwas eintippen können. Geben Sie eine Fahrzeugbezeichnung ein, z.B. Golf GT, und bestätigen Sie mit der *[Enter]*-Taste. Daraufhin wechselt das Eingabefeld zum Modelljahr.

Das Eingabefeld *Modelljahr* erwartet nun wieder die Eingabe einer Jahreszahl ab 1889. Wenn Sie versuchen, etwas anderes einzugeben, erhalten Sie beim Verlassen des Feldes mit *[Tab]* oder *[Enter]* eine Meldung. Wir werden in einem späteren Schritt darauf kommen, wie man solche Eingabeüberprüfungen festlegt.

Im Feld *Art* angelangt, stellen Sie fest, dass Sie hier nichts direkt eintippen können. Das liegt daran, dass Art ein Auswahlfeld ist. Wählen Sie mit *[Pfeil nach oben]* bzw. *[Pfeil nach unten]* einen Eintrag aus oder öffnen die *Werteliste* mit *[Alt]+[Pfeil nach unten]* oder einem Klick auf den Schalter.

Wenn Sie auch die Art mit *[Enter]* bestätigt haben, schaltet das Formular für Ihre weiteren Eingaben auf die zweite Seite um. Sie finden auf dem Formular alle vom Tabellenfenster bekannten Eingabe- und Auswahl-Felder sowie die *Markierungsschalter* für *Servo*, *ABS* und *Katalysator* wieder, die mit der linken Maustaste oder der Leertaste betätigt werden.

Zum Umschalten auf die dritte Seite haben wir diesmal einen speziellen, mit Verkauf beschrifteten Schalter eingebaut. Solche Schalter starten Makros, wenn sie betätigt werden, in diesem Fall den Befehl *SeiteAnzeigen(3)*. Wir kommen später darauf zurück.

Die dritte Seite bietet eine weitere Neuigkeit, die Sie entdecken, sobald Sie im Feld *Brutto-Preis* etwas eintragen und mit *[Enter]* bestätigen. Der *Netto-Preis* passt sich automatisch an, und zwar auch dann, wenn der *Rabatt-Satz* geändert wird. Sie sehen also jederzeit den korrekt berechneten Endpreis für das Fahrzeug in Ihrem Formular. Außerdem können Sie *Netto-Preis* und *Rabatt-Satz* nur ändern, wenn Sie das Häkchen für *Rabatt-Gewährung* gesetzt haben. Bei den Feldern auf dieser Seite werden Ereignisse und kurze Makros genutzt, aber das alles wird im Abschnitt *Schritt 13: Die Dateneingabe kontrollieren* noch genauer besprochen.

Nach der Eingabe für *Verkäufer* haben Sie zwei Möglichkeiten. Entweder Sie bestätigen auch hier mit *[Enter]*, wodurch automatisch ein neuer leerer Datensatz erzeugt und zur Neueingabe angezeigt wird oder Sie beenden die Neueingabe, indem Sie den Neueingabe-Schalter aus der Arretierung lösen bzw. *Bearbeiten/Neue Datensätze eingeben* aus dem Menü wählen.

Sollten Sie aus Versehen die [Enter]-Taste betätigt und somit überflüssigerweise einen neuen Datensatz erzeugt haben, wählen Sie einfach *Bearbeiten/Rückgängig: Datensatz*. Nach einer Kontrollfrage wird der unnötige Datensatz entfernt und der vorherige angezeigt.

2.7 Schritt 6: Suchen und markieren

Wenn die Anzahl der Datensätze größer wird, geht der Überblick verloren und man kann ein bestimmtes Auto nicht mehr durch einfaches Blättern heraussuchen. Stattdessen benötigt man eine Suchfunktion, der man bestimmte Bedingungen vorgeben kann. *TurboDB Studio* bietet Ihnen hier einerseits eine sehr schnelle und einfache direkte Suche und andererseits eine komplexe Suche mit Suchbedingung und unterschiedlichen Aktionen.

Nehmen wir zunächst einmal an, Sie würden in der Datenbank nach einem Fahrzeug aus dem Jahr 1981 suchen. Dann wählen Sie unter *Suchen* den Menüpunkt *Nach Modelljahr* und tippen im Dialogfenster für die direkte Suche *1981* ein. Mit *Ok* sucht *TurboDB Studio* das erste Fahrzeug dieses Jahres in der Tabelle und selektiert den entsprechenden Datensatz. Der Befehl *Suchen/Weitersuchen* bzw. die *[F3]*-Taste bringen Sie dann jeweils zum nächsten Datensatz bezüglich der Sortierung nach dem Modelljahr. Zuerst sind das die anderen Autos des Jahrganges 1981, anschließend kommen die 82er usw.

Diese direkte Suchfunktion steht für all diejenigen Tabellenspalten und Spaltenkombinationen zur Verfügung, für die ein Index existiert. Ein Index ist ein sortiertes Inhaltsverzeichnis der Datensätze und kann jederzeit angelegt werden. Dies wird im Schritt sieben dieses Streifzuges im Kapitel *Schritt 7: Indexe* beschrieben.

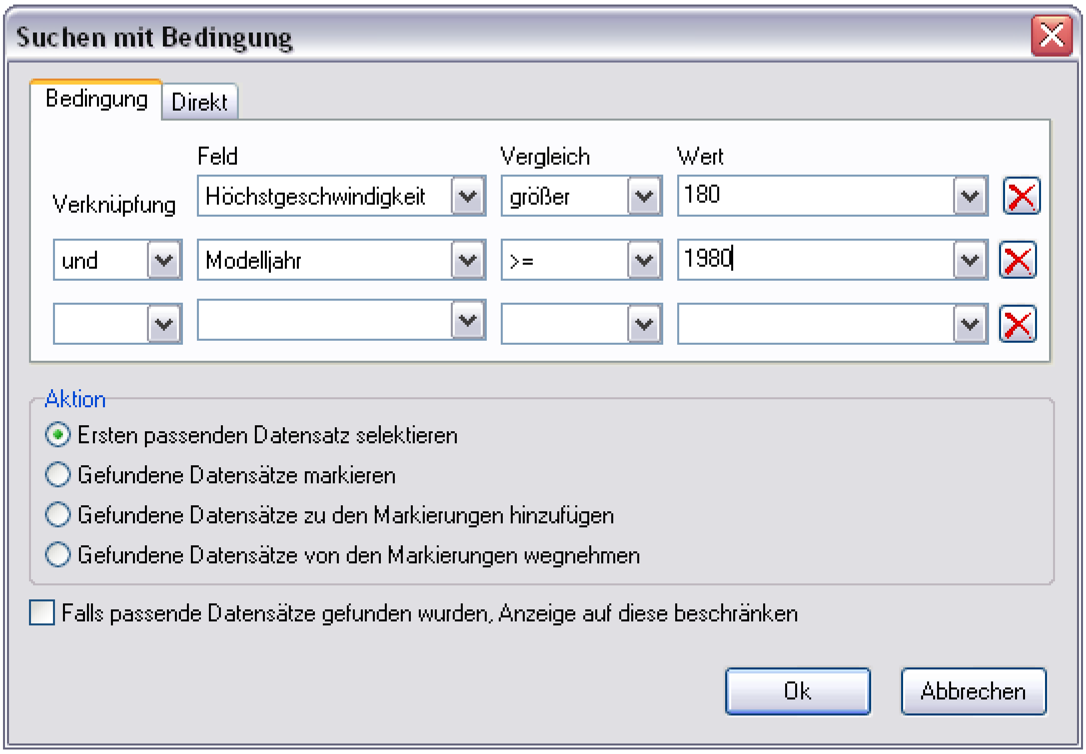


Der Such-Dialog für die Index-Suche: Einfach und schnell

Neben der direkten Suche können Sie auch wesentlich komplexere Suchbedingungen angeben und die gefundenen Datensätze nicht nur anzeigen, sondern auch markieren lassen. Als Beispiel interessieren wir uns für ein Fahrzeug, das mehr als 180 km/h Spitzengeschwindigkeit erreicht.

Mit dem Befehl Suchen/Mit Bedingung... erscheint ein Dialogfenster, in dem Sie Suchbedingung und Suchaktion eingeben können. Die Bedingung lautet in unserem Fall *Höchstgeschwindigkeit größer 180 und Modelljahr >= 1980* . Mit Hilfe der Auswahlboxen ist es kinderleicht, diese

Suchbedingung zu formulieren. In den Boxen mit der Überschrift *Feld* werden alle Datenfelder der Tabelle angeboten und unter *Vergleich* sind alle verfügbaren Vergleichsoperationen aufgeführt. Lediglich für den *Wert* müssen Sie noch selbst Hand an die Tastatur legen.
Als Suchaktion müssen Sie die Option *Ersten passenden Datensatz selektieren* aktivieren und das Häkchen bei *Anzeige auf gefundene Datensätze beschränken* entfernen. Die *[Enter]*-Taste startet die Suche und selektiert den gefundenen Datensatz.



Im Dialogfenster von "Suchen mit Bedingung" können Sie komfortabel Bedingungen für eine komplexe Suche zusammenstellen

Im nächsten Schritt soll erforscht werden, ob es auch von den Volkswagen-Werken ein so schnelles Auto gibt. Die Bedingung hierzu lautet *Hersteller wie "Volks*" und Höchstgeschwindigkeit > 180*. Der Operator *wie* in Kombination mit dem Sternchen * findet alle Einträge, die mit *Volks* beginnen. Das *und* bewirkt, dass nur solche Datensätze gefunden werden, in denen beide Bedingungen erfüllt sind. Eine solche Bedingung wird in *TurboDB Studio* als *Selektion* bezeichnet, weil dadurch bestimmte Datensätze selektiert werden. Auf einen bestimmten Datensatz bezogen, ist eine *Selektion* entweder erfüllt oder nicht erfüllt, d.h. wahr oder falsch. Eine genaue und ausführliche Beschreibung der Selektionen finden Sie an anderer Stelle in diesem Buch.

Selektionen sind sehr mächtige Suchmöglichkeiten, da man beliebig viele Teilbedingungen über *und* und *oder* verknüpfen kann. Bis zu drei solcher Teilbedingungen kann man ja schon auf unkomplizierte Art und Weise direkt im Dialog miteinander kombinieren. Wenn man jedoch mehrere Bedingungen benötigt, so muss man diese direkt eingeben.

Jetzt wollen wir wissen, welche Autos 5 Sitze haben. Dabei interessiert uns aber nicht einfach das erste Auto mit fünf Sitzen aufsuchen, sondern alle vorhandenen Fünfsitzer. Öffnen Sie also nochmals mit *Suchen/Mit Bedingung...* den Suchdialog und wählen Sie bei *Aktion* das Feld *Alle passenden Datensätze markieren*, bevor Sie den Suchdialog bestätigen. Falls entsprechende Datensätze gefunden werden, müssen Sie sich jetzt entscheiden, ob nur die passenden, oder aber diese inmitten der anderen, mit einem Sternchen markiert, angezeigt werden sollen. Das Resultat erkennen Sie am besten in der Tabellensicht (*Ansicht/Formularsicht* oder Formular-Schalter): Alle gefundenen Datensätze sind markiert, d.h. durch ein Sternchen in der Markierungsleiste gekennzeichnet. Wenn Sie die *[F8]*-Taste betätigen oder mit den Datensatz bei gedrückter *[Strg]* -Taste anklicken, wird die Markierung des selektierten Datensatzes umgeschaltet. Mit dieser Markierung könnten Sie nun weiterarbeiten. Sie können z.B. - sofern Sie dies nicht schon vorher gefordert haben - nur die markierten Datensätze anzeigen (*Ansicht/Nur markierte Datensätze* bzw.

[F7]) oder alle markierten Datensätze löschen (*Bearbeiten/Markierte Datensätze löschen*). Vor allem aber können Sie den Ausdruck von Berichten auf die markierten Datensätze beschränken (z.B. *Ausführen/Begleitbrief*).

2.8 Schritt 7: Indexe

Ganz allgemein spielen Indexe im Zusammenhang mit Datenbanken eine sehr wichtige Rolle. Wenn Ihre Tabellen größer werden, dauert es immer länger, einen bestimmten Datensatz zu suchen, falls man dabei jedesmal von vorne alle durchgehen muss. Deshalb legt man ein sortiertes Inhaltsverzeichnis der Datensätze an, mit dessen Hilfe man den gesuchten sofort findet.

Ein Beispiel: In der Auto-Tabelle benötigt man oft das Modelljahr des Fahrzeugs. Aus diesem Grund legt man einen nach dem Modelljahr sortierten Index an. Der erste Eintrag in diesem Index verweist auf das neueste, der letzte auf das älteste Fahrzeug der Tabelle. Nun gibt es aber mehrere Autos mit dem gleichen Modelljahr, deren Reihenfolge untereinander nicht festgelegt ist. Deshalb kann man beim Aufbau eines Index mehrere Sortierkriterien angeben. Ein Index über Modelljahr, Hersteller und Bezeichnung würde somit den 1981er Alfa Spider von Alfa-Lancia vor dem 1981er 200 D von Daimler einsortieren und letzteren wiederum vor dem 250 D der gleichen Firma.

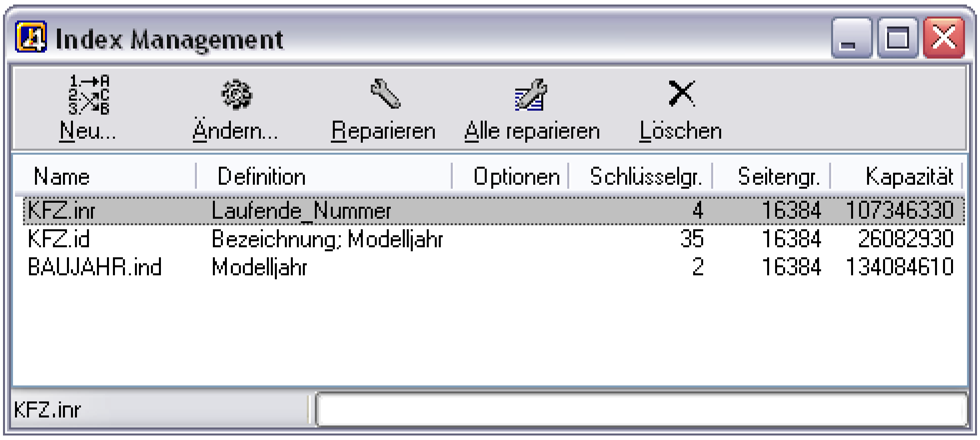


Abbildung:Mit dem Indexmanager verschaffen Sie sich einen schnellen Überblick über die existierenden Indexe

Aufgebaut ist ein solcher Index schnell. Öffnen Sie dazu im Projektnavigator das Datenmodell mit einem Doppelklick auf das Datenbanksymbol oder mit dem Schalter *Ausführen*. Klicken Sie dann auf die Tabelle für die Sie einen neuen Index erzeugen möchten. Mit dem Befehl *Tabelle/Indexe...* im Menü gelangen Sie zum Index-Manager. Hier finden Sie zunächst eine Übersicht aller Indexe, die für die selektierte Tabelle zur Zeit existieren. Auf der linken Seite dieser Liste steht der Name der Datei, in welcher der Index abgelegt ist. Danach folgt eine Beschreibung der Komponenten, aus denen der Index aufgebaut wurde. Die Reihenfolge dieser Komponenten spiegelt die Wichtigkeit der Sortierkriterien wieder, wobei die Priorität von links nach rechts immer weiter abnimmt. Es ist also etwas völlig anderes, ob Sie als Indexdefinition zuerst Modelljahr und dann Hersteller und Bezeichnung, oder erst Hersteller gefolgt von Modelljahr und Bezeichnung eingeben.

Mit dem Knopf *Neu...* oder *Ändern...* öffnen Sie einen Dialog, mit dessen Hilfe Sie einen neuen Index definieren können.

Auf der ersten Seite des Registers befindet sich ein Feld für den *Indexnamen*, also den Dateinamen ohne Dateierweiterung (*.ind). Im Optionsfeld für den *Indextyp* können Sie festlegen ob die Sortierung anhand von *Datenfeldern* oder anhand einer *Berechnungsvorschrift* bestimmt werden soll. Mit dem Markierungsfeld *Eindeutigkeit* können Sie zudem eindeutige Werte für neue Werte des indizierten Feldes oder der indizierten Felder erzwingen.

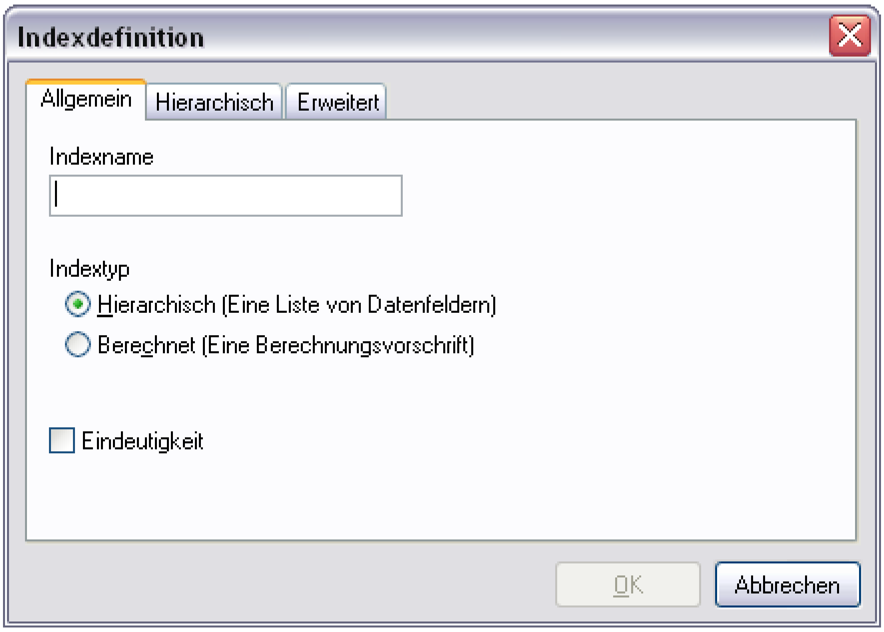


Abbildung: Definition eines neuen Index, Seite 1

Auf der zweiten Seite sehen Sie eine Liste aller Spalten der aktuellen Tabelle. Hier können Sie bestimmen, welche Komponenten, also Felder der Datentabelle, an der Sortierung beteiligt sein sollen. In der rechten Box, die zu Beginn noch leer ist, wird der Index zusammengebaut. Mit dem Knopf kann man nun selektierte Einträge der Spaltenliste zum Index-Aufbau hinzufügen. Mit kann man selektierte Index-Komponenten wieder löschen, und entfernt alle bisher eingefügten Index-Komponenten. Für jede Komponente kann außerdem eine Sortierreihenfolge festgelegt werden. Bei einer aufsteigenden Reihenfolge steht also der Eintrag mit dem niedrigsten Wert des Sortierkriteriums an erster Stelle, bei einer absteigenden entsprechend umgekehrt.

Wenn die gewählte Tabellenspalte ein sehr langer alphanumerischer Typ ist, empfiehlt es sich, nicht alle Zeichen zur Sortierung heranzuziehen, sondern z.B. nur die ersten 20. In diesem Fall tragen Sie vor dem Hinzufügen die gewünschte Länge in das Feld *Länge* ein. Diese Beschränkung hat den Vorteil, dass der Index schneller bearbeitet werden kann und weniger Platz auf Ihrer Festplatte beansprucht.

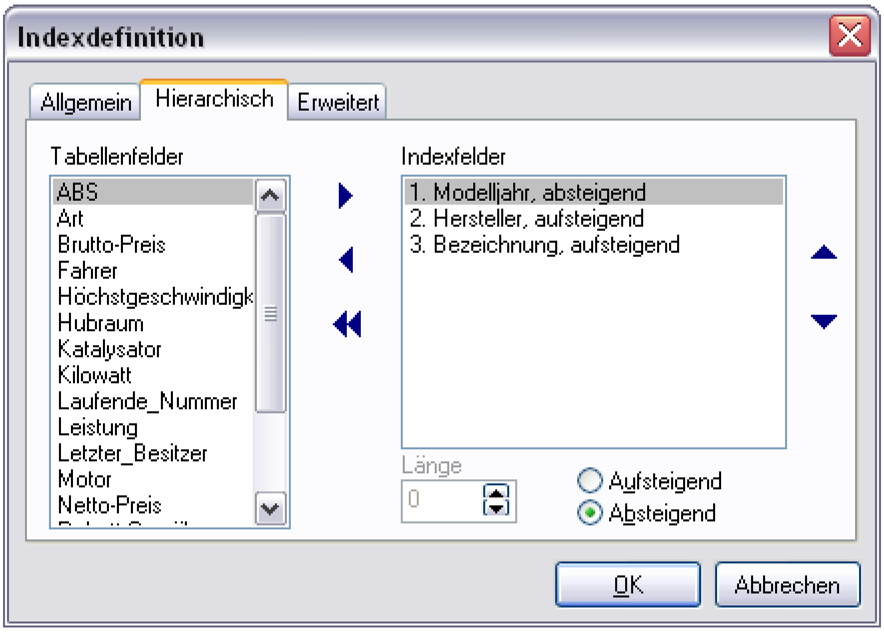


Abbildung: Definition eines neuen Index, Seite 2

Den Wert auf der dritten Seite belassen Sie am besten in der Standard-Einstellung.

Sobald Sie die Indexbeschreibung mit *OK* bestätigen, beginnt das Erstellen des Index. Dies dauert umso länger, je mehr Datensätze Ihre Tabelle enthält. Bei einigen hundert Datensätzen werden Sie kaum eine Verzögerung bemerken.

Wie können Sie nun feststellen, welchen Effekt der neue Index auf die Datenbank hat? Hätte Ihre Tabelle tausende von Einträgen, würden Sie deutlich erkennen können, dass die Suche z.B. nach Fahrzeugen, die vor 1985 gebaut worden sind, erheblich schneller abläuft. Bei einer kleinen Tabelle wie der in unserem Beispiel ist auch die normale Suche blitzschnell erledigt.

Allerdings können Sie den Index dazu verwenden, um sich die Datensätze in seiner Sortierreihenfolge anzeigen zu lassen. Sie brauchen nur *Ansicht/Sortierung...* zu wählen und erhalten eine Liste mit den Beschreibungen aller vorhandenen Indexe. Außer dem soeben von Ihnen erstellen Index über das Modelljahr finden Sie auch noch den gerade aktiven Index über die Satznummer sowie *Nur markierte Datensätze*, *Laufende_Nummer* und *Hersteller*.

[Unsortiert] steht für die Reihenfolge der Datensätze, wie sie in der Tabelle selbst vorliegt, und entspricht im großen und ganzen der Reihenfolge bei der Eingabe. Die neu eingegebenen Datensätze stehen hier also ganz am Ende. Auch *Nur markierte Datensätze* ist kein richtiger Index, sondern dient einfach dazu, nur die markierten Datensätze anzuzeigen. *Laufende_Nummer* ist ein von *TurboDB Studio* selbständig erzeugter Index, der die Einträge nach dem Inhalt der Spalte *Laufende_Nummer* sortiert. *Bezeichnung, Modelljahr* wurde erstellt, um die Suche nach bestimmten Fahrzeugen zu beschleunigen, und eine alphabetische Sortierung zur ermöglichen.

Selektieren Sie einfach Ihren soeben erstellten Index und bestätigen Sie die Eingabe. In der Tabellensicht sehen Sie am deutlichsten, dass die Fahrzeuge nun nach den von Ihnen bestimmten Kriterien geordnet sind.

Der Index-Manger bietet überdies die Möglichkeit, einen berechneten Index zu erstellen. Doch dazu mehr im Kapitel über *TurboDB Indexe*.

Mit dem Punkt *Löschen* können Sie einen vorhandenen und nicht mehr benötigten Index löschen. Die Knöpfe *Reparieren* bzw. *Alle reparieren* ermöglichen Ihnen schließlich, einen bzw. alle Indexe neu aufzubauen. Diese beiden letzten Befehle sollten Sie eigentlich nicht benötigen. Es kann aber schon einmal vorkommen, dass der Index beim Importieren von Datensätzen oder bei anderen komplizierten Operationen beschädigt wird. Sie erkennen das daran, dass plötzlich Datensätze zu fehlen scheinen, oder Lücken in der Spaltendarstellung auftauchen.

2.9 Schritt 8: Formulare entwerfen

Mit dem Tabellenfenster besitzt man bereits eine adäquate Form zur Darstellung und Manipulation von Daten. Dennoch ist es wünschenswert, Daten in einer übersichtlicheren Art und Weise zu präsentieren. Tabellen bieten zwar einen ganzheitlichen Überblick über den Bestand einer Datentabelle. Meistens ist man jedoch lediglich am Inhalt eines einzigen Datensatzes interessiert. Vor allem bei der Neueingabe und Bearbeitung von Informationen - und das sind wohl die Hauptaufgaben eines Datenbanksystems - sollte man die Möglichkeit besitzen, Daten in einer aufbereiteten und natürlich strukturierten Form darzustellen.

All diese Anforderungen erfüllen die Formulare. Wie man ein solches Formular entwirft bzw. bearbeitet, erfahren Sie am Beispiel eines zweiten Formulars im Beispiel-Projekt. Es soll *KFZ-Eingabe* heißen und ist vergleichsweise klein und einfach.

Schließen Sie nun das Datenfenster und klicken Sie im Projektnavigator den Ordner *KFZ* mit der rechten Maustaste an. Wählen Sie aus dem Kontextmenü den Punkt *Neu/Formular/Mit allen Feldern.* Daraufhin erscheint der bekannte Datei-Speichern-Dialog, in der auch gleich alle im Projektordner vorhandenen Formular-Dateien anzeigt. Geben Sie den Namen des neuen Formulars *KFZ-Eingabe* ein und bestätigen Sie mit dem Schalter*Speichern*.

Nachdem die Datei angelegt wurde, wird das neue Formular im Formulareditor geöffnet. Wenn Sie ein Formular *mit allen Feldern* erstellt haben, so sind bereits zwei Steuerelemente für jede Spalte der zugehörigen Tabelle angelegt: Eine Beschriftung und ein Datenfeld, d.h. ein Steuerelement das mit dem Feld der Tabelle verknüpft ist und daher auch gleich den Inhalt dieses Feldes der Tabelle anzeigt.

Sie können dem Formular neue Elemente hinzufügen indem Sie aus der Steuerelement-Liste am unteren Rand des Bildschirms die Elemente einfach auf das Formular ziehen. Hier haben Sie wiederum die Wahl zwischen sogenannten ungebundenen Steuerelementen (*Standard-Elemente*) die (noch) nicht mit Datenfeldern verknüpft sind und den gebundenen *Tabellen-Elementen* auf der Registerseite mit dem Namen der zugehörigen Tabelle. Tabellen-Elemente sind bereits automatisch mit den entsprechenden Datenfeldern verknüpft.

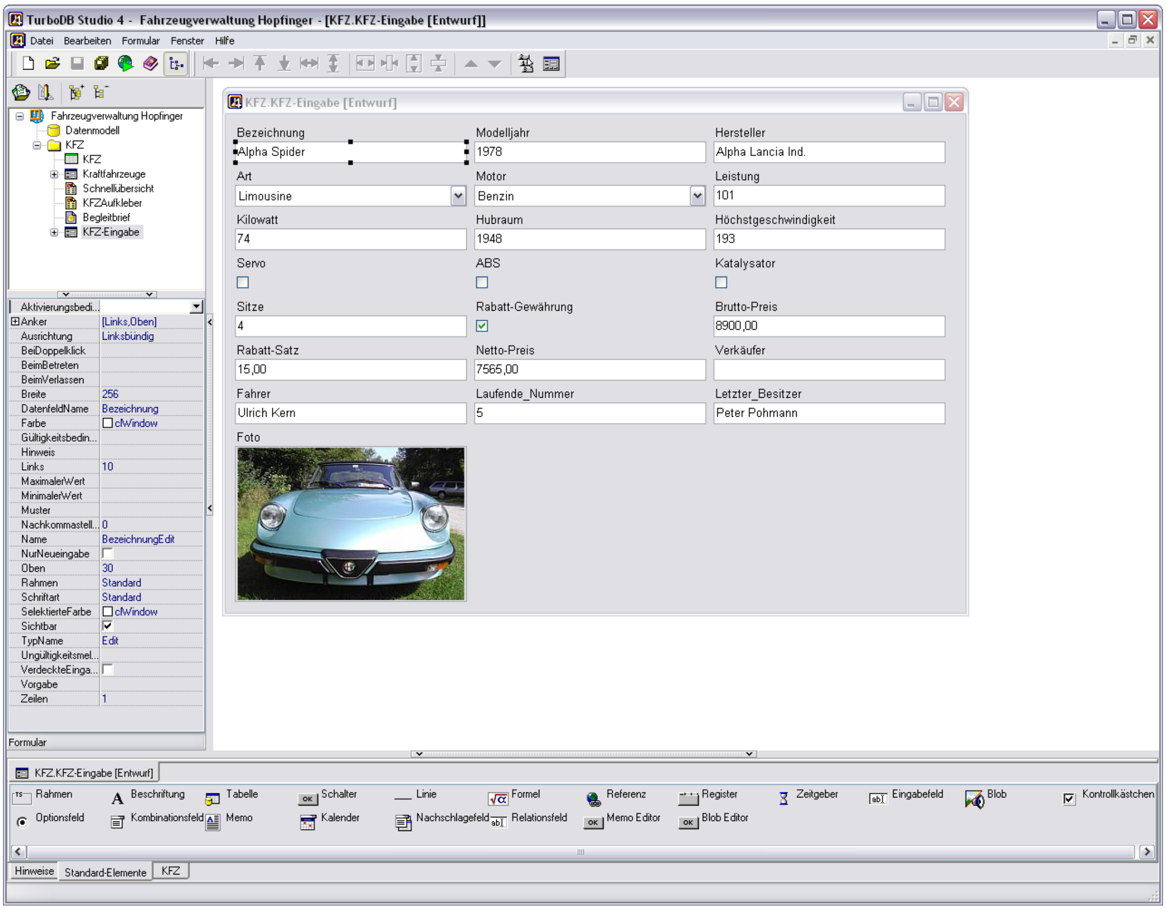


Abbildung: Der Formulareditor

Klicken Sie einmal auf eines der Felder. Acht quadratische Punkte, die Griffe, zeigen an, dass es selektiert ist. Sie können die Griffe mit der Maus bewegen, aufziehen und dadurch das Element verschieben oder es in der Größe verändern. Die Höhe von Feldern mit Text ist allerdings fest, da dies nur durch die Größe der verwendeten Schriftart und die Eigenschaft *Zeilen* bestimmt wird. Wenn Sie die Größe eines Beschriftungs- oder Eingabefeldes nach unten aufziehen, bleiben das Feld und seine Position gleich.

Sie können auch mehr als ein Feld selektieren, indem Sie bei gedrückter *[Umschalt]*-Taste noch ein anderes Element anklicken. Nun sind beide markiert, und die Griffe sind grau, da Sie sie nun nicht mehr ändern können. Eine dritte Möglichkeit der Selektion besteht darin, dass Sie bei *gedrückter [Strg]- und linker Maustaste* ein Rechteck über die gewünschten Elemente aufziehen.

Wenn mehr als ein Element selektiert ist, sehen Sie, dass in der Schalterleiste die Schalter zum Ausrichten der Steuerelemente aktiviert werden. Mit ihnen kann man die Steuerelemente aneinander ausrichten und ihre Größen anpassen. Außerdem können Sie mehrere markierte Steuerelemente gleichzeitig mit der Maus an eine andere Position verschieben. Die Mehrfachauswahl können Sie entfernen indem Sie ein nicht markiertes Steuerelement anklicken oder einfach die *[Esc]*-Taste drücken.

Neben seiner Position und Größe hat jedes Element noch viele weitere Eigenschaften, die auf den Seiten seines zugehörigen Notizbuches eingesehen und verändert werden können.Wir wollen diese Eigenschaften einmal exemplarisch an dem Datenfeld *Bezeichnung*, das sich direkt unter der gleichnamigen Beschriftung befindet, näher betrachten. Durch einen Klick auf dieses Element markieren wir es, wodurch seine Eigenschaften (in alphabetischer Reihenfolge) im *Objektinspektor* auf der linken Seite unterhalb des Projektnavigators angezeigt werden.

Das *Aussehen* des Steuerelements können Sie mit folgenden Eigenschaften beeinflussen:

Anker	Diese Eigenschaft können Sie aufklappen und für jeden Rand des Steuerelements einen Anker setzen oder entfernen. Wenn der Anker gesetzt ist, dann ändert sich die Größe des Steuerelements in diese Richtung proportional zur Größenänderung des Formulars. Sie können das am Formular Kraftfahrzeuge nachvollziehen: Wenn Sie dessen Größe verändern, passen sich die Steuerelemente automatisch an die Größe des Formulars an.
Farbe	Hier können Sie die Hintergrundfarbe des Steuerelements festlegen. Beachten Sie, dass diese Einstellung nicht beachtet wird, wenn unter Rahmen die Einstellung Standard gesetzt ist und Sie als Betriebssystem WindowsXP mit XP-Stil betreiben (siehe Rahmen).
Rahmen	Hier können Sie das Aussehen des Rahmens festlegen. Wenn Sie als Rahmentyp Standard wählen, wird die Darstellung des Rahmens Ihrem Betriebssystem überlassen. Wenn Sie mit Windows XP arbeiten und den XP-Grafikstil aktiviert haben, so werden alle Steuerelemente gemäß dem eingestellten XP-Thema gezeichnet. Die Farbgebung der Steuerelemente (Hintergrundfarbe und selektierte Farbe) bestimmt dabei ebenfalls das Betriebssystem.
Selektierte Farbe	Legt die Farbe fest, die das Steuerelement haben soll wenn es selektiert ist.
Sichtbar	Wird hier das Häkchen entfernt, ist das Steuerelement nur noch im Formulareditor sichtbar.
Breite, Höhe, Links, Rechts	Mit diesen vier Eigenschaften legen Sie die Größe und die Position des Steuerelements auf dem Fomular fest.
Hinweis	Der Text, den Sie hier eintragen, wird als Hilfetext angezeigt, wenn der Mauszeiger ein paar Sekunden über dem Feld verweilt.

Folgende Eigenschaften sind für das *Format* der angezeigten Daten zuständig:

Ausrichtung	Diese Eigenschaft legt die Textausrichtung innerhalb des Steuerelements fest, d.h. der Text wird je nach Einstellung am linken oder rechten Rand oder aber in der Mitte dargestellt.
Muster	Mit der Eigenschaft *Muster* legen Sie das Format der Eingabe fest. Das Format für eine Bankleitzahl würde beispielsweise so aussehen: 00 000 000 Mehr dazu finden Sie unter Dateneingabe kontrollieren im Kapitel "Formulare gestalten".
Nachkommastellen	Hier können Sie die Anzahl der angezeigten Nachkommastellen einstellen. Die Datenbank speichert Fließkommazahlen übrigens immer mit maximal möglicher Genauigkeit.
Schriftart	Ruft den Schriftarten-Dialog auf (siehe unten).
Verdeckte Eingabe	Wenn Sie hier ein Häkchen setzen, werden anstatt des eingegebenen Textes nur Sternchen angezeigt. Auf diese Art können Sie beispielsweise eine Passwortabfrage realisieren.
Zeilen	Ein Textfeld kann auch mehrere Zeilen haben. Hier können Sie bestimmen wieviele.

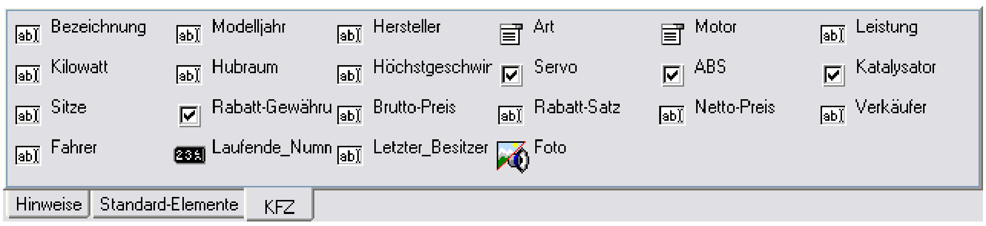
Besonders interessant sind die Eigenschaften *Gültigkeitsbedingung*, *MaximalerWert*, *MinimalerWert*, *NurNeueingabe*, *Ungültigkeitsmeldung* und *Vorgabe*. Mit ihnen können Sie festlegen, welche Werte unter welchen Bedingungen in Datenfelder eingegeben werden können. Auch diese Dinge werden im Abschnitt Schritt 13: Die Dateneingabe kontrollieren beschrieben.

Auf die Aktionen, die mit den Ereignissen *BeimBetreten*, *BeiDoppelklick* und *BeimVerlassen* ausgelöst werden können, soll erst in einem der späteren Kapitel eingegangen werden.

Doch zunächst zur Gestaltung des Formulars. Vielleicht merken Sie jetzt, dass die Anzahl der Felder auf dem automatisch erstellten Formular doch zu üppig ausgefallen ist, und einige Daten in diesem Formular gar nicht von Interesse sind. Deshalb sollen nun als erstes alle überflüssigen Felder gelöscht werden, so dass nur noch *Bezeichnung*, *Hersteller* und *Modelljahr* übrig bleiben. Am einfachsten erreichen Sie das, indem Sie durch Aufziehen eines Rechtecks mit der linken Maustaste (bei gedrückter *[Strg]*-Taste) möglichst viele Felder selektieren und dann *Bearbeiten/Löschen* wählen. Nach einer Sicherheitsabfrage werden alle selektierten Elemente gelöscht. Eventuell müssen Sie den Formulareditor noch etwas nach unten rollen, um alle Elemente löschen zu können.

Anschließend können Sie das Formular verkleinern, indem Sie es anklicken und dann den Griff in der Mitte des unteren Randes mit der Maus nach oben schieben.

Wollen Sie, dass die Ausführung eines Fahrzeuges im Formular erscheinen soll? Kein Problem. Das Feld Art kann man ganz einfach aus der Element-Palette einfügen. Diese befinden sich, wie schon eingangs erwähnt, in Form von eines Registers am unteren Rand des Rahmenfensters. In unserem Beispiel gibt es also einen Registerreiter *Standard-Elemente* und einen Registerreiter *KFZ*. Beim Inhalt der Registerseite mit dem Tabellennamen, also *KFZ* in diesem Fall, handelt es sich um die Palette aller Datenfelder dieser Tabelle. *Standard-Elemente* sind die Elemente, die unabhängig von Tabellenfeldern auf dem Formular platziert werden können. Man kann sie entweder von Hand mit Tabellenfeldern verknüpfen oder aber die in ein nicht verknüpftes Steuerelement eingegebenen Daten mit Makros abfragen und weiterverarbeiten.



In der Palette mit den Tabellenelementen finden Sie alle Felder der Tabelle als fertig vorkonfigurierte Steuerelemente

Wählen Sie also in der Element-Palette die Seite *KFZ*, klicken Sie auf *Art* und halten Sie die

Maustaste gedrückt. Ziehen Sie das Element per Maus an die gewünschte Stelle im Formular und lassen Sie den Maus-Button los. Beachten Sie, dass sich der Mauszeiger in ein Parkverbot-Schild verwandelt, wenn Sie das Element über einen Bereich ziehen, in dem das Element nicht abgelegt werden darf. Nun wollen wir das Element noch entsprechend ausrichten, damit unser Formular auch gleichmäßig aussieht. Mit der Maus alleine klappt das zumeist nicht beim ersten Mal. Selektieren Sie dazu Felder, die bündig untereinander angeordnet werden sollen. Wählen Sie dann *Formular/Position/Links* bzw. *Rechts* oder drücken Sie den Schalter bzw. in der Schalterleiste - je nachdem nach welcher Seite die Ausrichtung erfolgen soll. Dadurch werden alle selektierten Felder so verschoben, dass ihre linke Seite mit der linken Seite des am weitesten links angeordneten Steuerelements übereinstimmt.

Jetzt brauchen wir noch eine Beschriftung, damit man im Formular auch weiß, um welches Datenfeld es sich beim eben eingefügten handelt. Schalten Sie also in der Element-Palette auf die Seite *Standard-Elemente* um, wählen Sie *Beschriftung* aus, positionieren Sie das Element neben oder über dem neuen Datenfeld und richten Sie es entsprechend aus.

Den bisherigen Text des Elementes können wir nicht verwenden. Im Eigenschaften-Bereich können Sie bei der Eigenschaft *Text* den neuen Beschriftungstext eingeben, also z.B. *Art*.

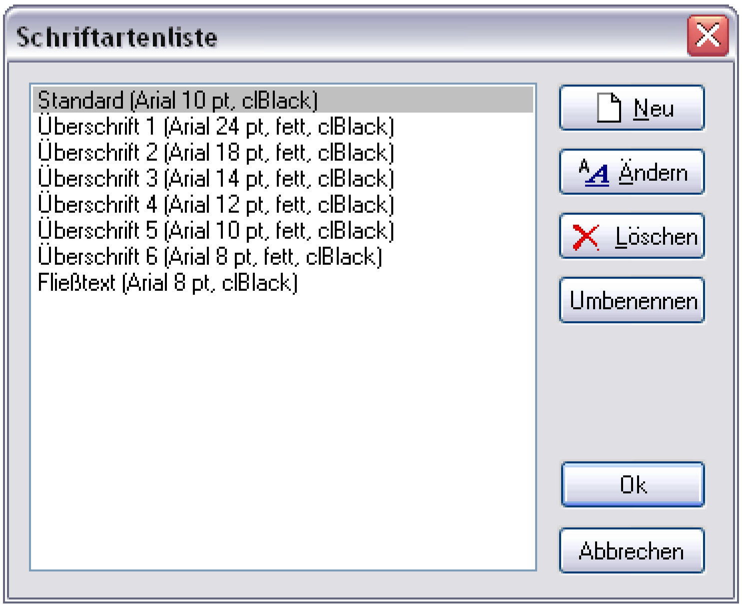


Bei Beschriftungen reicht es meist, die Eigenschaft Text auszufüllen

Vielleicht gefallen Ihnen Schriftart und -größe des Formulartitels nicht. Werfen wir also einmal einen Blick auf die Eigenschaft *Schriftart*. Klicken Sie zuerst die Eigenschaft an und rufen Sie dann mit einem Klick auf den im Feld rechts erscheinenden Schalter den Schriftartendialog auf. Das Dialogfenster enthält eine Liste mit den vorhandenen Schriftarten, die Sie sofort verwenden können.

Sie können die Einträge in der Liste aber auch ändern, indem Sie auf einen Eintrag klicken und danach auf den Schalter *Ändern*. Beachten Sie bitte, dass sich eine Änderung auf alle Felder im Formular auswirkt, die diese Schriftart verwenden. auf diese Weise kann man sehr schnell das Aussehen eines Formulares verändern.

Sie können der Liste auch neue Schriftarten hinzufügen indem Sie auf den Schalter *Neu* klicken.



Im Schriftarten-Dialog können Sie die im Formular verwendeten Schriftarten zentral verwalten

Vielleicht erscheint Ihnen diese Vorgehensweise wie ein Umweg gegenüber einer direkten Schriftenauswahl für jedes Element. Es ist jedoch so, dass erfahrungsgemäß nur wenige Schriftarten gleichzeitig in einem Formular benutzt werden. Das Zuweisen der gleichen Schrift an viele Maskenelemente geht über eine Schriftartenliste erheblich effektiver und ist weniger fehleranfällig.

Schließen Sie nun den Formulareditor und beantworten Sie die Frage nach dem Abspeichern mit Ja. Nun genügt ein Doppelklick auf das neue Formular im Projektnavigator, um es öffnen.

2.10 Schritt 9: Mehrere Tabellen

Projekte mit nur einer Tabelle sind eine eher langweilige Angelegenheit. Interessant wird es, wenn man mit mehreren verknüpften Tabellen arbeitet. Aus diesem Grund enthält unser Beispiel noch eine zweite Tabelle mit Namen und Adressen von Käufern der Fahrzeuge. Die Datei dieser Tabelle heißt *Kunden.dat* und ist ganz einfach in das bestehende Projekt zu integrieren.

Schließen Sie das Datenfenster und wählen Sie *Datei/Projektelement hinzufügen...* oder einfach *Hinzufügen/Tabelle...* im Kontextmenü des Projektnavigators. Im bereits bekannten Dateiauswahldialog selektieren Sie die Datei *Kunden.dat* und bestätigen die Eingabe. Daraufhin wird ein weiterer Ordner für die zusätzliche Tabelle an das Ende des Projektes angefügt. Der Ordner enthält auch gleich die neue Tabelle.

Mit einer Tabelle alleine kann man zwar schon einiges anfangen, doch wie bereits gesagt mehr Übersicht bei der Datenpflege bietet ein Formular. Und genau dieses ist wiederum im Projektverzeichnis unseres Beispiels zu finden und kann mit *Hinzufügen/Formular...* leicht eingebunden werden. Vergewissern Sie sich zuvor, dass im Projektnavigator der neu hinzugefügte Ordner *Kunden* oder eines der darin enthaltenen Projektelemente selektiert ist. Wählen Sie dann im Dateiauswahldialog unter Dateityp *TurboDB Formular (*.frm)* aus und öffnen Sie die Datei *Kunden.frm*. Nun erscheint das Formular-Symbol mit dem Titel *Kunden* am Ende des Projektes. Wenn Ihnen dieser Titel nicht gefällt, ändern Sie ihn in den Eigenschaften im im Objektinspektor.

Öffnen Sie mit einem Doppelklick das hinzugefügte Formular der Tabelle *Kunden* und schalten Sie mit *Bearbeiten/Datensätze ändern* den Eingabemodus ein. Hier möchten wir Ihre Aufmerksamkeit auf das Feld *Fahrzeug* lenken. Die Einträge in diesem Feld werden Ihnen bekannt vorkommen; es sind Verweise auf die Tabelle *KFZ*, und sie geben an, welchen Fahrzeugtyp der entsprechende Kunde erworben hat.

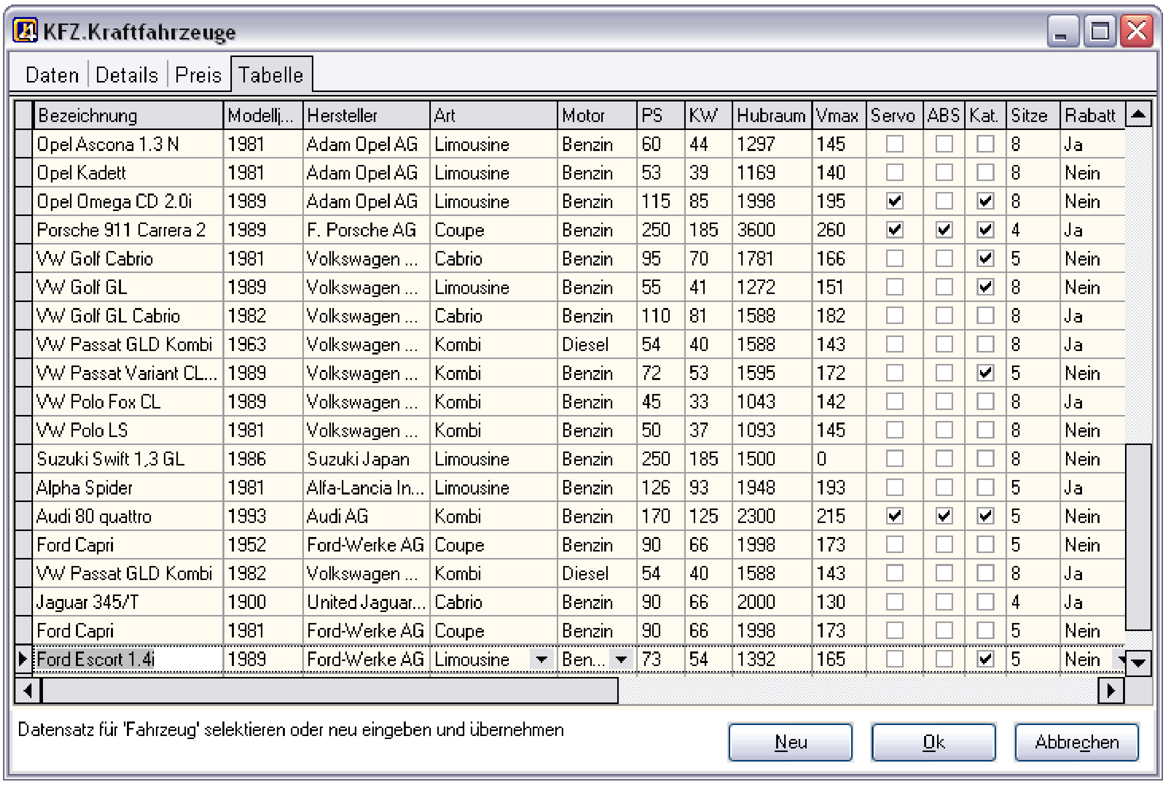
Das Datenfeld Fahrzeug ist allerdings kein gewöhnliches Datenfeld. Es handelt sich um einen speziellen Feldtyp zum Herstellen von Verknüpfungen zwischen Tabellen. Sein Name ist *Koppelfeld*, weil es einen Datensatz einer anderen Tabelle an einen Datensatz der eigenen Tabelle ankoppelt. Koppelfelder sind Bestandteil eines innovativen Systems zur Automatisierung

von Tabellenverknüpfungen, dem ADL-System (*Automatic Data Link*). Die Funktionsweise dieses Systems möchten wir Ihnen nun anhand des Feldes *Fahrzeug* demonstrieren.

Fokussieren Sie dieses Feld und drücken Sie dann *[F4]* oder den Schalter ... . Dadurch öffnet sich ein zweites Datenfenster, diesmal für die Tabelle KFZ. Der angekoppelte Datensatz (falls einer angekoppelt war) ist selektiert. Durch ein spezielles Bedienfeld am unteren Fensterrand weist sich das Datenfenster als ADL-Fenster aus. Ganz links finden Sie die Information: *Datensatz für 'Fahrzeug'*. Datensatz selektieren und übernehmen. Daneben die drei Schalter *Neu*, *Ok* und *Abbrechen*. Selektieren Sie einen beliebigen Datensatz und klicken Sie auf Ok. Das ADL-Fenster wird geschlossen und im Feld Fahrzeug des Ausgangs-Formulars steht der neue Eintrag.

Dieser Mechanismus ist nur dann aktiviert, wenn das Datenfenster auf *Datensätze ändern* oder *Neue Datensätze eingeben* geschaltet ist.

Nun wollen wir es mal andersherum versuchen. Das Feld Fahrzeug hat einen weißen Hintergrund, Sie können also etwas eintippen. Versuchen Sie z.B. *Fiat* und bestätigen Sie mit *[Enter]* Ein Fiat ist in der Tabelle *KFZ* noch nicht enthalten, deshalb öffnet sich wieder das ADL-Fenster von *KFZ* mit der Meldung *Eintrag für 'Fiat' selektieren oder neu eingeben und übernehmen.* Drücken Sie auf den Schalter Neueingabe, und ein neuer Datensatz wird angelegt. Tragen Sie in diesen die Informationen für den neuen Fahrzeugtyp in die entsprechenden Felder ein. Wenn alle Angaben eingetragen sind, betätigen Sie den Schalter *Ok*, um das ADL-Fenster zu schließen. Ihr neuer Fiat findet sich nun im Feld *Fahrzeug* des Formulars *Kunden*. Wie Sie sehen, nimmt Ihnen das ADL-System bei der Eingabe in verknüpfte Tabellen so viel Arbeit ab, wie es einem Programm nur irgend möglich ist.



Bezeichnung	Modellj...	Hersteller	Art	Motor	PS	KW	Hubraum	Vmax	Servo	ABS	Kat.	Sitze	Rabatt
Opel Ascona 1.3 N	1981	Adam Opel AG	Limousine	Benzin	60	44	1297	145	☐	☐	☐	8	Ja
Opel Kadett	1981	Adam Opel AG	Limousine	Benzin	53	39	1169	140	☐	☐	☐	8	Nein
Opel Omega CD 2.0i	1989	Adam Opel AG	Limousine	Benzin	115	85	1998	195	☑	☐	☑	8	Nein
Porsche 911 Carrera 2	1989	F. Porsche AG	Coupe	Benzin	250	185	3600	260	☑	☑	☑	4	Ja
VW Golf Cabrio	1981	Volkswagen ...	Cabrio	Benzin	95	70	1781	166	☐	☐	☑	5	Nein
VW Golf GL	1989	Volkswagen ...	Limousine	Benzin	55	41	1272	151	☐	☐	☑	8	Nein
VW Golf GL Cabrio	1982	Volkswagen ...	Cabrio	Benzin	110	81	1588	182	☐	☐	☐	8	Ja
VW Passat GLD Kombi	1963	Volkswagen ...	Kombi	Diesel	54	40	1588	143	☐	☐	☐	8	Ja
VW Passat Variant CL...	1989	Volkswagen ...	Kombi	Benzin	72	53	1595	172	☐	☐	☑	5	Nein
VW Polo Fox CL	1989	Volkswagen ...	Kombi	Benzin	45	33	1043	142	☐	☐	☐	8	Ja
VW Polo LS	1981	Volkswagen ...	Kombi	Benzin	50	37	1093	145	☐	☐	☐	8	Nein
Suzuki Swift 1,3 GL	1986	Suzuki Japan	Limousine	Benzin	250	185	1500	0	☐	☐	☐	8	Nein
Alpha Spider	1981	Alfa-Lancia In...	Limousine	Benzin	126	93	1948	193	☐	☐	☐	5	Ja
Audi 80 quattro	1993	Audi AG	Kombi	Benzin	170	125	2300	215	☑	☑	☑	5	Nein
Ford Capri	1952	Ford-Werke AG	Coupe	Benzin	90	66	1998	173	☐	☐	☐	5	Nein
VW Passat GLD Kombi	1982	Volkswagen ...	Kombi	Diesel	54	40	1588	143	☐	☐	☐	8	Ja
Jaguar 345/T	1900	United Jaguar...	Cabrio	Benzin	90	66	2000	130	☐	☐	☐	4	Ja
Ford Capri	1981	Ford-Werke AG	Coupe	Benzin	90	66	1998	173	☐	☐	☐	5	Nein
Ford Escort 1.4i	1989	Ford-Werke AG	Limousine	Ben...	73	54	1392	165	☐	☐	☑	5	Nein

Abbildung:Ein ADL-Fenster ist ein modales Datenfenster mit einem zusätzlichen Informations-Panel zur Verknüpfung von Datensätzen

Neben dem Koppelfeld, das ja immer nur einen Datensatz der anderen Tabelle ankoppeln kann, gibt es noch Relationsfelder, die beliebig viele Verknüpfungen herstellen können. Solche Felder eignen sich besonders gut für Stichwortlisten in Literaturdatenbanken und werden im Kapitel *Die Automatic Data Link Technologie* ausführlich beschrieben.

Für den Augenblick möchten wir nur noch einen kurzen Blick auf die Funktionsweise des ADL-Systems werfen. Wenn Sie kein Interesse an der Theorie haben, können Sie den Rest dieses Abschnitts überspringen. Falls Sie dagegen etwas mehr wissen wollen, starten Sie nun den *Tabellen Designer* für die Tabelle *Kunden* über den Menüpunkt *Entwerfen* des Kontextmenüs das erscheint, wenn Sie das Tabellensymbol rechtsklicken oder einem Klick auf die Tabelle gefolgt von einem Klick auf den *Entwerfen*-Schalter im Projektnavigator. Auch wenn Sie keine Änderungen an der Tabelle vornehmen wollen, ist dieses Dialogfenster eine gute Möglichkeit, den Aufbau von Tabellen zu untersuchen. Er enthält auf der linken Seite eine Liste aller Tabellenspalten.

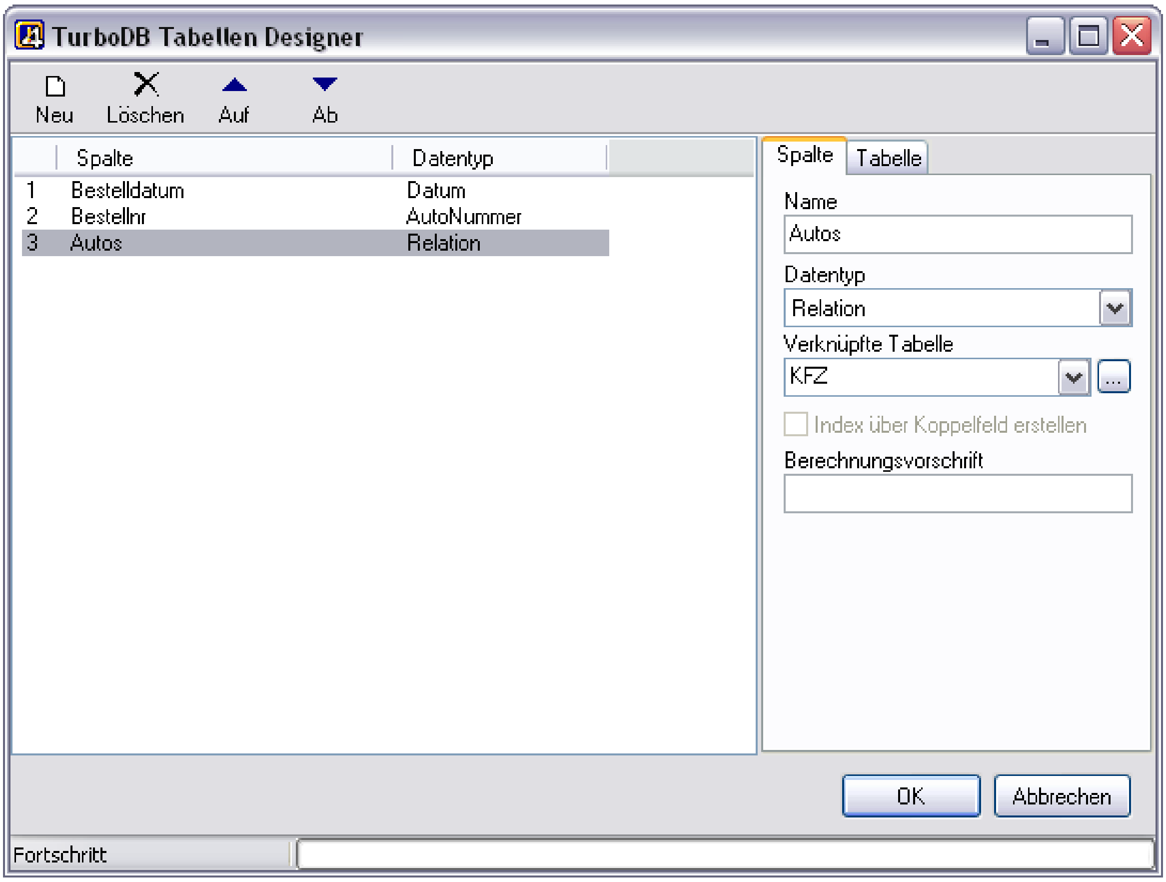
Selektieren Sie hier den Eintrag *Fahrzeug*. Auf der rechten Seite sehen Sie die Beschreibung einer Kopplungs-Spalte. Sie hat wie alle Spalten einen Namen und als einzige weitere Information den Namen der angekoppelten Tabelle: *KFZ*. Indem Sie *Abbruch* drücken, verlassen Sie den *Tabellen Designer* ohne Änderungen.

Die Tabelle *KFZ* wiederum trägt Ihren Teil zur Kopplung durch die Spalte *Laufende_Nummer* bei. Der Inhalt dieser speziellen Spalte ist eine Zahl zwischen eins und zwei Milliarden und wird von *TurboDB Studio* für jeden Datensatz ermittelt und automatisch eingetragen. Vor allen Dingen ist dabei gewährleistet, dass jeder Datensatz eine andere Nummer trägt. Diese Nummer wiederum wird im Koppelfeld der Tabelle *Kunden* abgespeichert um die Verknüpfung zwischen Kunde und Auto herzustellen. Damit das Koppelfeld im Formular *Kunden* keine für den Benutzer nichtssagende Zahl darstellt, enthält die Definition der Spalte *Laufende_Nummer* in der Tabelle *KFZ* noch die Angabe *Anzeige in verknüpften Tabellen*. Hier wurde festgelegt, dass das Koppelfeld die Bezeichnung und das Modelljahr des angekoppelten Fahrzeuges anzeigen soll.

2.11 Schritt 10: Eine neue Tabelle anlegen

Wenn Sie nach diesem Streifzug durch *TurboDB Studio* daran gehen, Ihre eigenen Projekte aufzubauen, werden Sie gleich zu Beginn vor einer Aufgabe stehen, die bisher noch nicht angesprochen wurde. Sie müssen eine neue Tabelle erstellen mit den passenden Spalten für die von Ihnen zu verwaltenden Informationen. Nehmen wir also an, Sie hätten Ihr eigenes Projekt mit *Datei/Neu/Projekt...* schon erstellt und möchten nun eine Tabelle anlegen. Wir sprechen das am Beispiel einer Tabelle mit den Bestellungen des Händlers für das KFZ-Projekt durch.

Als erstes klicken Sie im Projektnavigator das Datenmodell mit der rechten Maustaste, wählen aus dem Kontextmenü *Neu/Tabelle...* und geben im Dateiauswahldialog *Bestellung.dat* als Dateiname für die neue Tabelle an. Anschließend erscheint der *Tabellen Designer*, in dem Sie den Aufbau der neuen Tabelle festlegen. Der Dialog umfasst zwei Bereiche: Der linke Bereich stellt eine Liste der Tabellenspalten dar, der rechte Bereich ist zum Ändern der Spalteneigenschaften da. Der Eigenschaften-Bereich besteht aus zwei Registerseiten, aber davon soll uns vorerst nur die Erste interessieren. Sie sehen am oberen Rand vier Schalter mit denen sie Spalten hinzufügen, entfernen und ihre Reihenfolge verschieben. Da die Liste links alle schon definierten Spalten der Tabelle enthält, ist sie momentan noch leer.



Im Tabellen Designer werden die Spalten der Tabelle definiert

Beginnen wir mit einer Spalte für das Bestelldatum. Klicken Sie auf den Schalter *Neu*. Eine neue Spalte wird der Liste hinzugefügt. Geben Sie rechts ins Feld *Name* den einen Namen für die Spalte ein (z.B. Bestelldatum) und selektieren Sie im Auswahlfeld darunter den Eintrag *Datum*.

Um eine Bestellnummer zu vergeben, greifen Sie am besten auf die automatische Nummerierung von *TurboDB Studio* zurück. Fügen Sie mit *Neu* der Liste eine neue Spalte vom Typ *Auto-Nummer* hinzu. Diese enthält dann für jeden Datensatz eine automatisch vergebene Nummer, die garantiert nicht doppelt auftaucht.

Nun gibt es noch die Möglichkeit, zu bestimmen, in welcher Form von einer verknüpften Tabelle aus auf die Bestellung verwiesen wird. Geben Sie hier Bestellnr ein. Die Option *Eindeutig*, mit der für dieses Feld nur eindeutige Einträge zulassen werden, ist damit natürlich hinfällig, da die Bestellnummer vom Typ Auto-Nummer und damit sowieso eindeutig ist. Man könnte aber auch auf die Idee kommen, die Auto-Nummer als *Bestelldatum* anzeigen zu lassen. Da jedoch an einem Tage hoffentlich mehrere Bestellungen eingehen, könnten diese dadurch nicht mehr eindeutig identifiziert werden. Würde man nun jedoch auf der Eindeutigkeit für das Feld *Bestelldatum* bestehen, könnte pro Tag nur eine Bestellung aufgenommen werden, was wohl ziemlich realitätsfremd ist.

Als nächstes geht es darum, die bestellten Autos mit der Bestellung zu verknüpfen. Ein Koppelfeld reicht hierzu nicht aus, weil ein Autohändler ja normalerweise mehrere Fahrzeuge eines Typs auf einmal bestellt. Dies ist ein Fall für das Relationsfeld. Es gleicht in seiner Funktionsweise dem Koppelfeld, nimmt aber beliebig viele angekoppelte Datensätze auf. Beim Hinzufügen der Spalte Fahrzeuge vom Typ Relation müssen Sie auch nicht mehr als Spaltenname und angekoppelte Tabelle angeben.

Damit wäre eine einfache Tabelle für Bestellungen entworfen. Bestätigen Sie mit *OK*, um sie zu erstellen. Nach dem Erstellen der Tabelle sollten Sie auch gleich noch ein *Formular/mit allen Feldern...* erzeugen. Dieses wird auch gleich im Entwurfsmodus geöffnet und diese Tatsache sollten Sie gleich dazu nutzen, das Relationsfeld mit der Eigenschaft Zeilen etwas höher zu machen (z.B. *Zeilen*: 5) und bei der Eigenschaft *NachschlageSortierung* den Wert *KFZ.inr* auszuwählen. Damit werden die Daten im ADL-Panel nach Modellbezeichnung sortiert.

Nun können Sie anfangen, Bestellungen einzugeben (Formular im Projektnavigator doppelklicken und dann *Bearbeiten/Neue Datensätze eingeben* wählen). Dabei werden die Einträge unter Fahrzeuge ganz ähnlich wie beim Koppelfeld unterstützt. Selektieren Sie das Relationsfeld und drücken Sie die *[F4]*-Taste. Daraufhin zeigt Ihnen VDP das ADL-Fenster für KFZ. Nun markieren Sie die gewünschten Einträge durch *[Strg]+Klick* oder durch Drücken der *[F8]*-Taste und übernehmen die markierten Datensätze mit *Ok*. Die markierten Datensätze werden in das Feld Fahrzeuge eingefügt.

Die andere Art der ADL-Eingabe funktioniert ebenfalls. Tippen Sie einfach die gewünschten Fahrzeuge untereinander in das Relationsfeld. Beim Bestätigen mit *[Enter]* haben Sie dann die Gelegenheit, selektierte Datensätze zu übernehmen oder Ihre Eingabe als Grundlage für einen Datensatz zu verwenden.

Wie unter Windows allgemein üblich, verwendet *TurboDB Studio* die Tastenkombination *[Strg]+[Enter]* für den Zeilenvorschub innerhalb eines mehrzeiligen Eingabefeldes wie z.B. *Fahrzeuge*. Durch bloßes Drücken von *[Enter]* gelangen Sie ja in das nächste Datenfeld.

Bei der Eingabe in Datum-Felder wie z.B. *Bestelldatum* genügt ein *[H]* für Heute, um das aktuelle Datum einzutragen. Ebenso verstehen Zeit-Felder *[J]* für Jetzt.

Nachdem Sie einige Zeit mit der neuen Tabelle gearbeitet haben, werden Sie evtl. feststellen, dass ihre Struktur noch nicht ganz optimal ist. Vielleicht haben Sie ein Datenfeld vergessen, oder ein anderes ist zu kurz geraten oder der Zahlenbereich ist zu klein oder, oder... Glücklicherweise ist das kein großes Problem. Starten Sie einfach noch einmal den Tabellendesigner und nehmen Sie die notwendigen Änderungen vor.

Der *Tabellen Designer* hat auf der rechten Seite noch eine weitere Registerseite mit der Überschrift Tabelle. Hier können Sie die Tabelle vor unberechtigtem Zugriff schützen, indem Sie ein *Passwort* und optional einen *Verschlüsselungs-Code* eingeben. Falls Sie schon ein Passwort vergeben haben, können Sie es dort auch ändern bzw. aufheben. Außerdem können Sie hier das Dateiformat der Tabellendateien festlegen. Die aktuelle Version ist *Tabellenversion 3*, was für neue Tabellen auch voreingestellt ist. *Tabellenversion 2* entspricht dem Dateiformat des *TurboDB Studio*-Vorgängers *Visual Data Publisher 3*. DOS-kompatible Tabellen aus DOS-TDB 5.5 sowie WinTDB 1.0 bis VDP 2.0 haben Tabellenversion 1.0. Normalerweise sollten Sie hier die Voreinstellung belassen.

Bevor die Struktur der Tabelle wirklich modifiziert wird, erhalten Sie eine Liste der geplanten Änderungen. Die sollten Sie sich sehr genau ansehen. Sollten Sie nämlich im Struktureditor Spalten gelöscht haben, gehen die darin gespeicherten Informationen verloren. Achten Sie deshalb besonders auf Einträge der Form *Feldname gelöscht*. Dieser Fall tritt auch dann auf, wenn Sie eine Spalte löschen und dann eine gleichnamige Spalte mit einem unvereinbaren Datentyp einfügen. Etwas weniger gefährlich ist es, wenn Sie die Länge einer alphanumerischen Spalte herabsetzen. Zu lange Einträge werden beim Anpassen abgeschnitten. Wenn die aufgelisteten Änderungen Ihrer Absicht entsprechen, bestätigen Sie sie mit *Ok* und die Restrukturierung der Tabelle beginnt. Wenn schon sehr viele Datensätze eingetragen sind, ist das eventuell eine langwierige Operation, über deren Fortschreiten Sie durch einen Fortschrittsbalken informiert werden.

2.12 Schritt 11: Einen Bericht erstellen

Zur Ausgabe Ihrer Datensätze in aufbereiteter Form hält TurboDB Studio grundsätzlich zwei verschiedene Möglichkeiten bereit. Eine davon sind Berichte, die - ganz ähnlich wie Formulare - in einem graphischen Editor bearbeitet werden können. Sie können die verschiedensten Farben und Schriftarten enthalten. Auch Bilder sind für diese Berichte kein Problem. Die Alternative ist textorientiert und wird in den weiteren Schritten beschrieben.

Nebenbei sei bemerkt, dass die Ausgabe von Daten sich nicht auf das Drucken beschränkt, selbst wenn dies die Hauptanwendung darstellen mag. Demnach stehen weitere Alternativen zur Auswahl, wie zum Beispiel das Abspeichern eines Berichts als DOS-Text oder im HTML-Format.

Wir wollen nun einen Bericht für die Fahrzeuge entwerfen, genauer gesagt eine Liste des Fuhrparks. Selektieren Sie dazu den Ordner *KFZ* im Projektnavigator und drücken Sie die rechte

Maustaste um das Kontextmenü aufzurufen. Klicken Sie hier auf den Menüpunkt *Neu/Bericht...*. Nachdem Sie im Dateiauswahl-Dialog die Datei für den neuen Bericht angelegt haben, startet der Bericht-Editor. Vom Prinzip her funktioniert er ganz ähnlich wie der Formulareditor. Seine Funktionalität weist jedoch einige Unterschiede auf, denn jetzt geht es ja in erster Linie darum, die Daten "zu Papier zu bringen".

Hinter dem Schlagwort Bericht verbergen sich viele Arten von Darstellungsformen - egal, ob es sich dabei um Etiketten, Briefumschläge, Serienbriefe oder Listen handelt. In unserem Beispiel entscheiden wir uns für eine tabellarische Liste, aber dazu später mehr.

Zunächst sollten werfen wir einen Blick auf das Hauptmenü, welches durch das Öffnen des Berichteditors um den Menüpunkt *Bericht* angereichert wurde.

Wählen Sie im Menü den Punkt *Bericht/Seiten-Eigenschaften...*, was einen neuen Dialog öffnet. Hier werden Angaben zu den Ausmaßen einer Seite gemacht, auf der die Daten untergebracht werden sollen. Da ja meistens auf Seiten genormter Größen gedruckt wird, sind die gebräuchlichsten in der Auswahlbox links erhältlich. Eine manuelle Eingabe sollte also lediglich bei exotischen Formaten notwendig sein. Wenn die Ausgabe auf einer um 90 Grad gedrehten Seite erfolgen soll, wählen Sie einfach die Option *Querformat*.

Nun fehlt nur noch das Festlegen der Seitenränder auf der dritten Seite. Bedenken Sie, dass alle Größenangaben in Millimeter gemacht werden müssen. Wenn Sie die Option *An Druckbereich anpassen* aktivieren, dann wird der Rand automatisch durch den Bereich, in dem Ihr Drucker drucken kann, festgelegt.

Mit dem Menüpunkt *Bericht/Bericht-Eigenschaften...* können Sie einen Filter, *Selektion* genannt, für die zu druckenden Daten festlegen. Außerdem können Sie die Sortierung der Daten bestimmen.

Mit *Bericht/Liste der Datenfelder anzeigen...* können Sie schließlich noch das Datenfelder-Fenster einblenden, welches alle im Projekt enthaltenen Tabellen und ihre Felder enthält. Mit Hilfe dieses Fensters können Sie die Datenfelder auswählen, die im Bericht erscheinen sollen indem Sie die Felder einfach auf den Bericht ziehen.

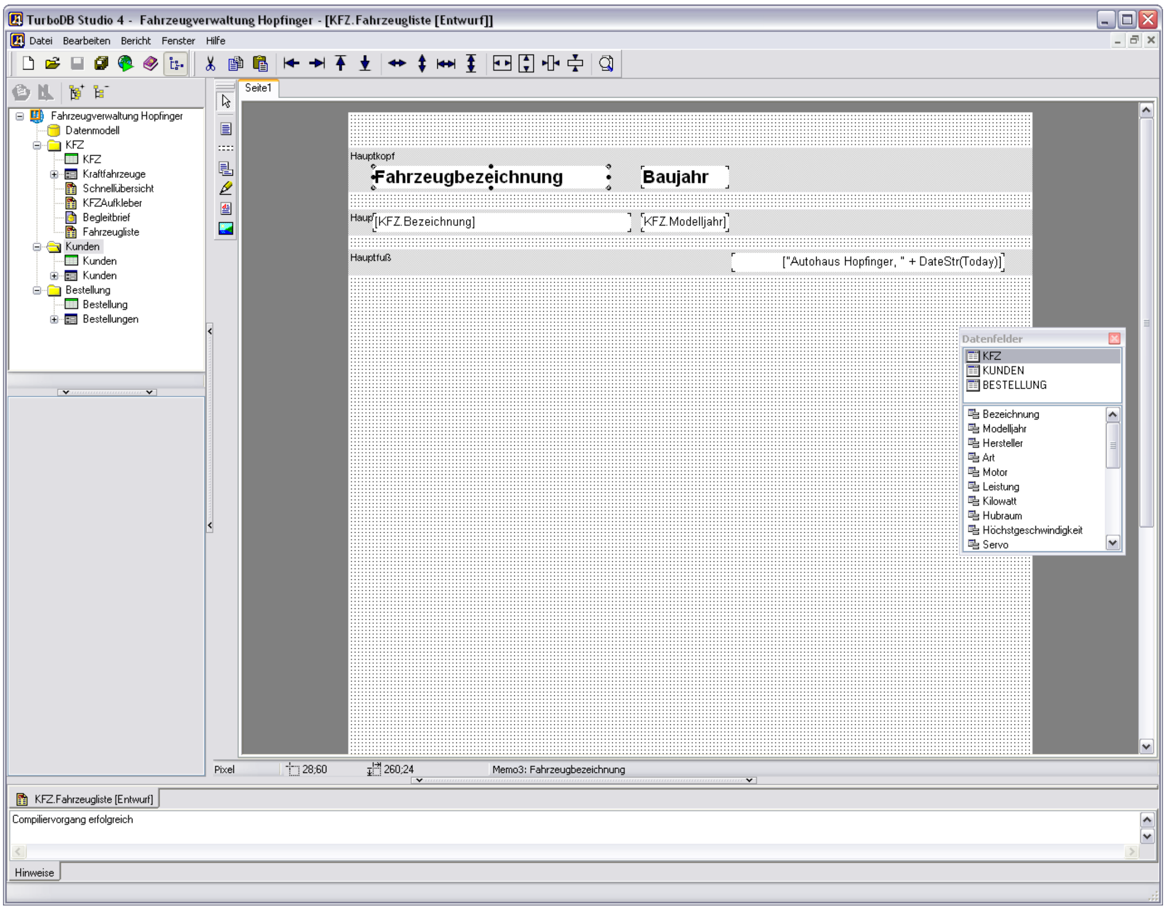


Mit dem Datenfelder-Fenster können Sie die Daten Ihrer Tabellen mit "Ziehen und Ablegen" bequem in den Bericht einfügen

Doch zunächst sollten Sie Ihre Aufmerksamkeit auf die Palette am linken Rand des Berichteditors richten. Hier gibt es verschiedene Elemente, wovon aber momentan nur die ersten beiden interessant sind: Klicken Sie auf den Schalter *Band einfügen* und anschliessend auf den Bericht, um einen Bereich (auch Band genannt) einzufügen. Eine Auswahlliste mit verschiedenen Bereichen erscheint. Wählen Sie hier *Hauptkopf* aus und wiederholen Sie den Vorgang danach um noch einen einen *Hauptdaten*-Bereich einzufügen.

Um den Bericht mit Daten zu füllen, ziehen Sie nun die Felder *Bezeichnung* und *Modelljahr* aus dem *Datenfelder*-Fenster auf den *Hauptdaten*-Bereich. Die Felder sind automatisch mit den Datenfeldern verknüpft wenn Sie sie aus der *Liste der Datenfelder* auf den Bericht ziehen. Klicken Sie nun noch *Feld einfügen* in der Palette an, um ungebundene Felder für die Beschriftung auf

dem *Hauptkopf*-Bereich zu platzieren. Ändern Sie den Inhalt des Feldes indem Sie die Eigenschaft *Formel* anwählen und dann den Schalter klicken. In dem erscheinenden Editor-Fenster können Sie den anzuzeigenden Text eingeben. In unserem Fall wären das *Fahrzeugbezeichnung* bzw. *Baujahr*.



Der Bericht-Editor ähnelt dem Formulareditor, enthält aber verschiedene Bereiche für einen Bericht

Im Unterschied zu Formularen kann ein Bericht verschiedenste Bereiche enthalten. Die Liste der verfügbaren Bereiche haben Sie ja schon gesehen. Die Funktion dieser verschiedenen Bereiche besteht in der Gliederung des Berichtes. Der *Report Titel* wird nur einmal ganz zu Beginn des Berichtes ausgedruckt und ist deshalb der geeignete Platz für Überschriften, den Namen des Autors, Ort, Datum usw. Dementsprechend wird die *Report Zusammenfassung* nur am Ende des Berichtes gedruckt und enthält meistens Zusammenfassungen, Summen usw. Der *Hauptdaten* -Bereich wird für jeden Datensatz ausgedruckt, deshalb enthält er die eigentlichen Datenfelder. *Hauptkopf*- und *Hauptfuß*-Bereich sind oben und unten auf jeder Seite zu finden. Es gibt noch jede Menge anderer Bereiche, die aber im Kapitel Bericht-Bereiche näher erläutert werden.

Jetzt könnten Sie z.B. noch ihre Schriftart ändern, indem Sie bei der Eigenschaft *Schriftart* eine neue Schriftart und Ihre Darstellung aus den entsprechenden , von Windows bekannten, Listen auswählen.

Natürlich können Sie jederzeit in noch weitere Felder aus der Elementpalette einfügen, beispielsweise ein zusätzliches Feld einbauen, das die Firma und das Druckdatum ausgibt. Schreiben Sie dazu in den Formeleditor den Ausdruck

```
["Autohaus Hopfinger, " + DateStr(Today)]
```

Da es sich hierbei um einen berechneten Ausdruck handelt, müssen zu Beginn und am Ende des Ausdrucks eckige Klammern angegeben werden. Statischen Text können Sie direkt (auch ohne Anführungszeichen) in die Felder schreiben.

Mit dem Berichteditor kann man auch Serienbriefe gestalten. Das Vorgehen ist dabei das selbe

wie beim Erstellen dieses Beispielberichts. Der Unterschied liegt darin, dass bei einem Serienbrief pro Datensatz eine komplette Seite gedruckt werden muss. Zu diesem Zweck gibt es bei den Bericht-Bereichen eine Eigenschaft *Seitenvorschub*. Diese bewirkt, dass nach dem Drucken dieses Bereiches eine neue Seite begonnen wird. Für Serienbriefe ist es deshalb durchaus sinnvoll, auf die Bereichsvielfalt zu verzichten und alle Daten im Hauptbereich auszugeben.

Eine andere Besonderheit von Serienbriefen ist es, dass die Daten üblicherweise in ein Anschreiben in Form eines mehr oder weniger langen Textes eingebettet sind. Das bringt die Schwierigkeit mit sich, dass man nicht einfach ein Datenfeld für die Ausgabe der Inhalte der Datenbank benutzen kann, es sei denn man formatiert den Bericht entsprechend. Da dies aber nicht immer sinnvoll ist, gibt es auch noch die Möglichkeit, Daten eingebettet in einen Text auszugeben.

Zunächst brauchen wir einen neuen Bericht, dieses mal für die Tabelle *Kunden*. Mittlerweile wissen Sie ja schon wie das geht: Einfach den Ordner *Kunden* anklicken, mit der rechten Maustaste das Kontextmenü aufrufen und *Neu/Bericht...* wählen und im Dateiauswahl-Dialog den Dateinamen *Serienbrief.ber* eintippen.

Klicken Sie wieder auf *Band hinzufügen* um einen *Hauptdaten*-Bereich zu erzeugen. Klicken Sie im *Datenfelder*-Fenster die Tabelle Kunden an und ziehen Sie die Felder *Name*, *Vorname*, *Straße*, *PLZ* und *Ort* in den Bericht. Zusätzlich zu diesen Feldern brauchen wir noch zwei ungebundene Felder aus der Palette: Eines für die Anrede und eines für den Text des Anschreibens.

Klicken Sie das Feld für die Anrede an und rufen Sie mit dem Schalter  der Formel-Eigenschaft den Formeleditor auf. Tragen Sie folgenden Ausdruck als Formel ein:

```
['Sehr geehrte' + CHOICE($Kunden.Geschlecht, 'r Herr ', ' Frau ') +
$Kunden.Name + ',']
```

Wie bereits oben erwähnt, bedeuten die eckigen Klammern, dass es sich nicht um einen Text handelt, sondern um einen berechneten Ausdruck oder ein Datenfeld. CHOICE ist eine TurboPL-Funktion und bewirkt, dass abhängig vom Inhalt des Datenfeldes Geschlecht entweder der erste Text ausgegeben wird oder der zweite.

Ganz ähnlich sieht dann auch die Formel für das Anschreiben aus:

```
nach unseren Unterlagen haben Sie kürzlich einen
[$Kunden.Fahrzeug.Bezeichnung] der Firma [$Kunden.Fahrzeug.Hersteller] bei uns
erworben.

Leider wurde für dieses Fahrzeug ein sicherheitskritischer Produktionsfehler
bekannt, der schnellstmöglich repariert werden muß.

Wenden Sie sich deshalb bitte umgehend an unsere Zentrale in Radstadt.



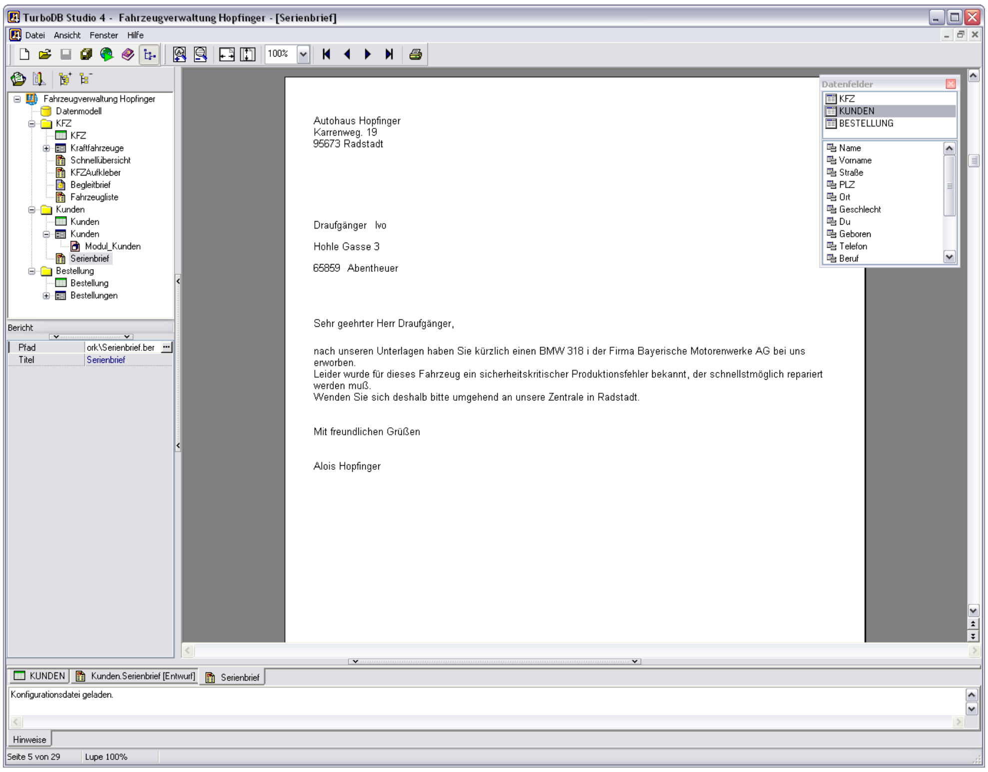
Mit freundlichen Grüßen



Alois Hopfinger
```

Wie Sie sehen, können Sie auch einfach den Text eintippen und, in eckigen Klammern eingefasst, die Datenfelder dazwischen einstreuen. Falls Ihnen die Feldbezeichnung *Kunde.Fahrzeug.Bezeichnung* seltsam vorkommt: Bei der Spalte *Fahrzeug* handelt es sich um ein Koppelfeld und Dank des *ADL-Mechanismus'* ist es möglich über diese sogenannte *Koppelfeld-Notation* auf alle Felder der verknüpften Tabelle zuzugreifen.

Während der Arbeit an dem Bericht, können Sie im Menü Zusätze jederzeit einen Probeausdruck auf dem Drucker oder in der Druckvorschau produzieren. Alternativ dazu können Sie das Druckvorschausymbol der Schalterleiste oder die Taste *[F9]* verwenden. So sehen Sie immer sofort, wie sich Änderungen auswirken und müssen nicht "auf Verdacht" arbeiten.

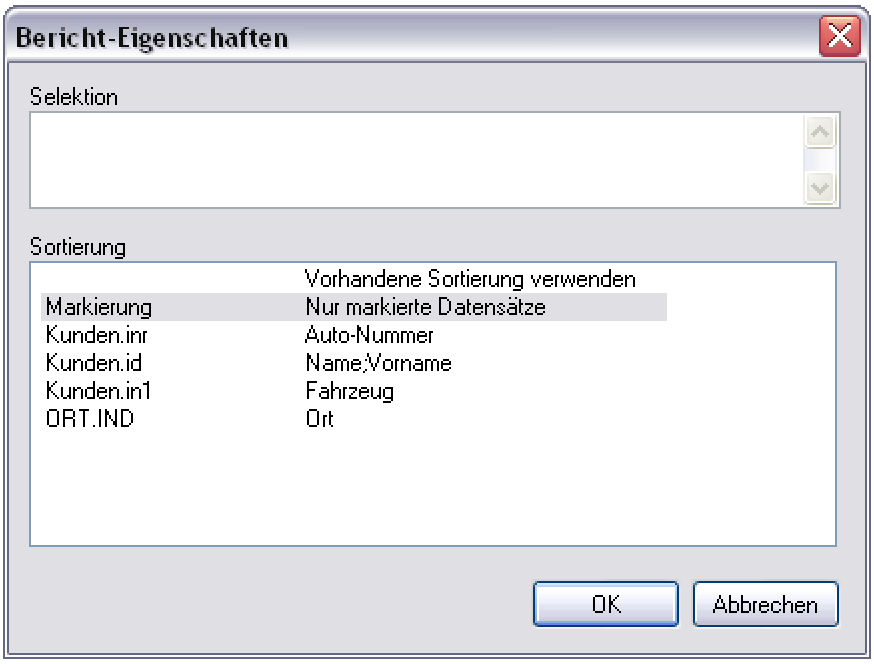


Auch während der Arbeit an Ihrem Bericht können Sie jederzeit einen Blick auf das Resultat werfen.

Wenn der Bericht fertig gestellt ist, können Sie ihn auch aus dem Projektnavigator heraus ausdrucken. Dazu selektieren Sie ihn und wählen *Ausführen* aus dem Kontextmenü. Ein *Doppelklick* mit der linken Maustaste oder ein Klick auf den Schalter *Ausführen* startet ebenfalls den Ausdruck.

Wenn Sie unseren Serienbrief in der Druckvorschau betrachten, dann wird ihnen vielleicht auffallen, dass auch Kunden angeschrieben werden, die gar kein Auto gekauft haben. Auch die Tatsache, dass die Kunden unabhängig von der Automarke und des Modells angeschrieben werden, entspricht nicht ganz der Praxis.

Beide Probleme können Sie ganz einfach lösen, indem Sie im Menü *Bericht/Bericht-Eigenschaften...* wählen um den Bericht-Einstellungen Dialog zu öffnen. Klicken Sie her die Zeile *Markierung* | *Nur Markierte Datensätze* an und Schließen Sie den Dialog mit *Ok*.



Im Bericht-Eigenschaften Dialog können Sie die Daten für den Bericht filtern und sortieren

Wenn Sie nun den Bericht in der Vorschau oder mit Doppelklick auf das Symbol im Projektnavigator starten, so wird kein einziger Brief gedruckt. Das kommt daher, dass der Bericht jetzt nur noch markierte Datensätze druckt und weder bei der Vorschau noch bei einem Start aus dem Projektnavigator eine passende Datengrundlage vorhanden ist.

Öffnen Sie daher das Formular *Kunden* mit einem Doppelklick auf das Symbol im Projektnavigator und schalten Sie mit dem Befehl *Ansicht/Formularansicht* auf die Tabellenansicht um. Sie können natürlich auch einfach den Registerreiter "Tabelle" anklicken. Sortieren Sie die Tabelle nach Fahrzeugen indem Sie unter *Ansicht/Sortierung...* oder im *Kombinationsfeld* in der Schalterleiste den Eintrag *Fahrzeug* auswählen.

Nun können Sie mit *[Strg]+Klick* alle Kunden markieren, die beispielsweise einen *VW Golf GL* von *1989* gekauft haben. Wählen Sie danach im Menü den Befehl *Ausführen/Serienbrief* um den Bericht namens Serienbrief auszuführen. Wenn Sie Ihren Bericht nicht wie oben vorgeschlagen *Serienbrief* benannt haben, dann steht im Menü Ausführen statt Serienbrief der Name ihres Berichts.

Natürlich haben Sie noch weitere Möglichkeiten, die Suche komfortabler zu gestalten, beispielsweise mit dem *Suchen mit Bedingung* Dialog, den Sie ja schon aus *Schritt 6: Suchen und markieren* kennen. An dieser Stelle sei nur noch auf das *Beispielprojekt KFZ* verwiesen, auf dem dieses Tutorial basiert. Dort gibt es in der Tabellenansicht der Formulare *KFZ* und *Kunden* Schalter, die verschiedene Lösungen für das gezielte Suchen und markieren (mit anschließendem Drucken) von Datensätzen zeigen.

2.13 Schritt 12: Serienbriefe mit OLE

Windows bietet Ihnen nicht nur eine ansprechende und einfach zu benutzende Oberfläche, sondern auch mehrere Möglichkeiten, Windows-Anwendungen automatisiert zu nutzen. *TurboDB Studio* unterstützt die sogenannte *OLE-Automatisierung* mittels *COM*, das auch als *ActiveX* bekannt ist. Das waren jetzt viele Schlagworte in schneller Folge, weshalb wir uns zuerst mal die Bedeutung der einzelnen Begriffe näher ansehen sollten:

Unter *OLE-Automatisierung* versteht man das 'fernsteuern' von Anwendungen aus der eigenen Anwendung heraus, beispielsweise könnten Sie *Excel* mit Ihrer *TurboDB Studio* Anwendung so steuern, dass es Tabellen anlegt, formatiert und auch gleich noch mit den Daten aus Ihren Tabellen füttert.

Um dies bewerkstelligen zu können, müssen Sie ein *TurboPL*-Programm schreiben das mit

OLE-Objekten arbeitet. *OLE* bedeutet *Object Linking and Embedding*, was im Programmierer-Jargon so viel heisst wie 'fremde Programme und Objekte im eigenen Code benutzen'. Das ganze funktioniert zwar nur mit *ActiveX-Objekten*, aber das sollte Sie momentan nicht weiter stören, da eigentlich alle größeren Anwendungen *ActiveX*-fähig sind. *ActiveX* ist ein Begriff aus der Microsoft'schen Marketingabteilung der bedeutet, dass die Anwendung oder die DLL-Datei *COM* unterstützt. *COM* steht übrigens für *Component Object Model* und ist nichts anderes als eine vereinheitlichte Schnittstelle für Objekte, damit man sie in Form von (COM-fähigen) EXE- oder DLL-Dateien mit einer beliebigen (COM-fähigen) Programmiersprache für eigene Zwecke benutzen kann.

Der langen Rede kurzer Sinn:
Wenn Sie Ihre Daten beispielsweise in Ihrer Tabellenkalkulation oder Ihrer Textverarbeitung benutzen möchten, dann haben Sie mit *TurboDB Studio* die Möglichkeit dazu.

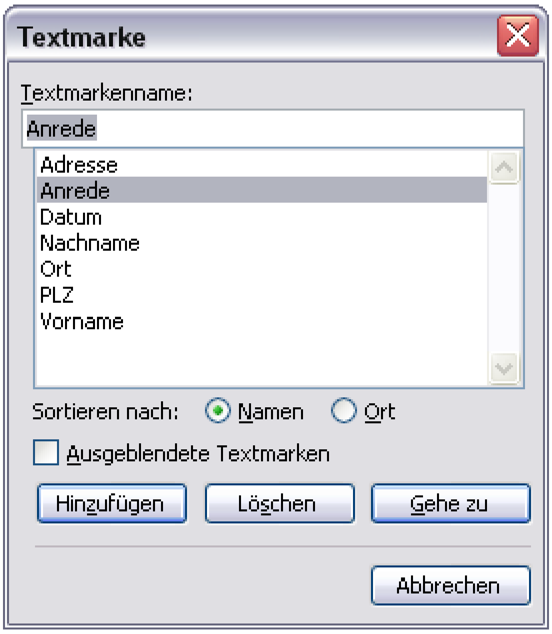
Eines sei allerdings sei gleich vorweggenommen: Das Thema OLE-Automatisierung hat durchaus seine Tücken und ist mehr für fortgeschrittene Anwender mit etwas Programmier-Erfahrung denn für Einsteiger geeignet. Dennoch wollen wir dieses Thema hier anschneiden, denn jeder fängt mal klein an und wir hoffen, dass wir Ihr Interesse wecken können. Wenn Sie in dieses Thema etwas tiefer einsteigen möchten, sehen Sie sich bitte das *Beispielprojekt OleDemo* an, dort finden Sie im Kommentar des Moduls auch noch wertvolle Tipps für den Einstieg in die OLE-Programmierung mit *TurboDB Studio*.

Um im Rahmen dieses Tutorials zu bleiben bedienen wir uns für dieses Beispiel dem mitgelieferten Modul *OleWord.mod*, das Sie im Installationsverzeichnis von *TurboDB Studio* im Unterverzeichnis *Lib* finden. Dieses Modul enthält einige Funktionen, die einfach zu benutzen sind und Ihnen die eigentliche Programmierung mit OLE-Objekten abnehmen. Fügen Sie dieses Modul dem Ordner Kunden hinzu.

☞ Unser Beispiel setzt voraus, dass Sie eine deutsche oder englische Version von Microsoft Word 2000 (oder eine neuere Version) auf Ihrem Computer installiert haben. OLE-Automatisierung würde zwar auch mit anderen Anwendungen funktionieren aber aufgrund der großen Verbreitung von Microsoft Word haben wir uns für ein Beispiel mit dieser Anwendung entschieden.

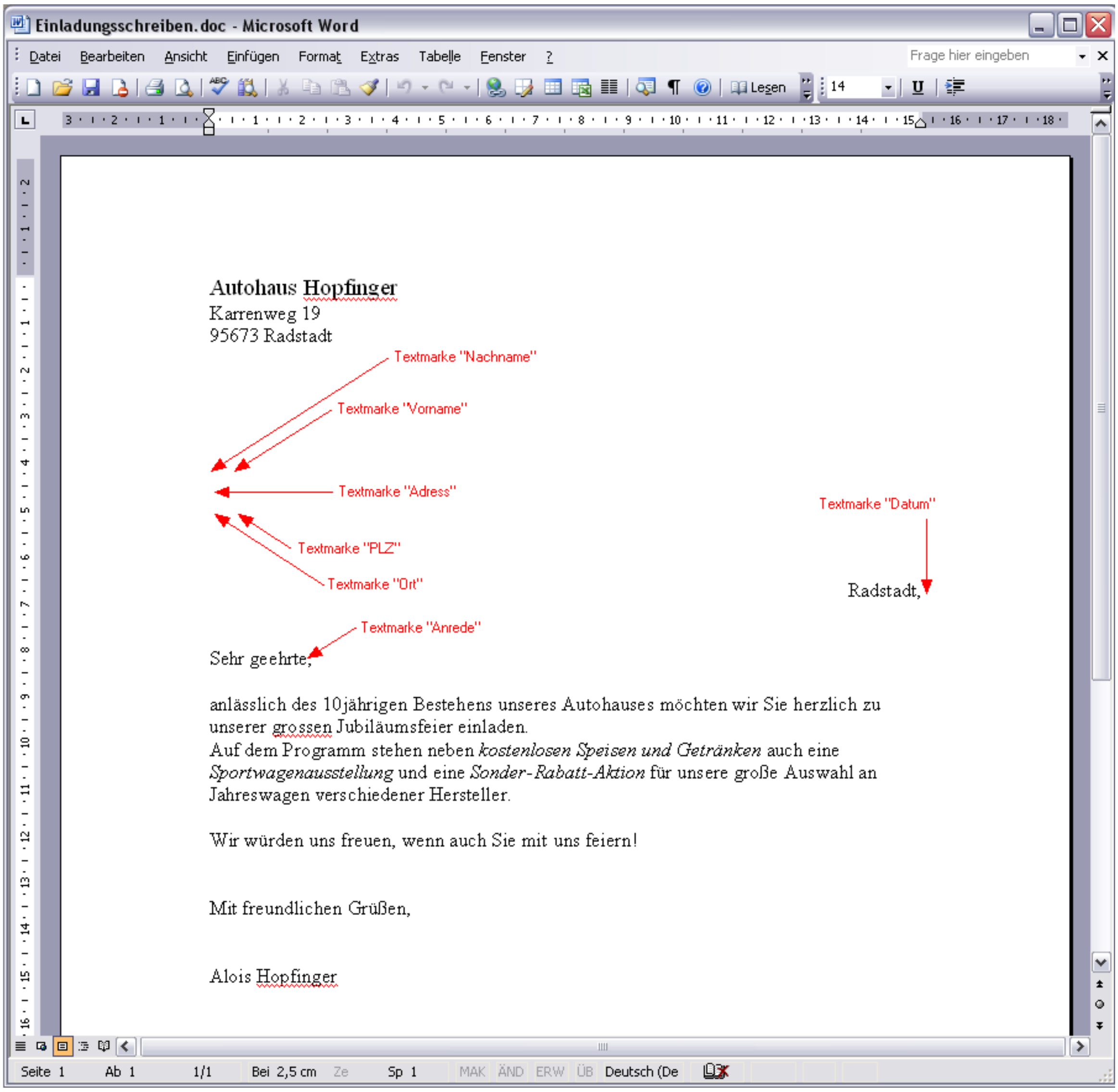
Für unseren Serienbrief mit Word benötigen wir zunächst eine Vorlage in Form eines Word-Dokuments, in das unser TurboPL-Programm dann seine Daten schreibt. Um die Daten zielgenau positionieren zu können, unabhängig vom restlichen Inhalt und der Formatierung, benutzen wir die sogenannten Textmarken (engl. "Bookmarks", siehe auch: VisualBasic-Hilfe von Word). Diese werden folgendermaßen definiert:

Starten Sie Word und laden Sie gegebenenfalls ein Dokument. Positionieren Sie den Cursor an die Stelle im Text, an der Sie später die Daten einfügen möchten. Wählen Sie dann im Menü *Einfügen/Textmarken...* (engl.: *Insert/Bookmark*) um den Textmarken-Dialog zu öffnen. Tragen Sie hier im Feld *Textmarkenname* (*Bookmark name*) den Namen ein, mit dem Sie die Textmarke später im TurboPL-Programm aufrufen möchten und fügen Sie mit dem *Hinzufügen*-Schalter (*Add* -Schalter) die Textmarke der Liste hinzu.



Mit den Textmarken von Word können Sie festlegen, an welche Stellen das TurboPL-Programm seine Texte später schreiben soll.

Als Beispiel erstellen wir ein Einladungsschreiben anlässlich des 10jährigen Bestehens der Firma. Schreiben Sie das Anschreiben so, wie Sie es gewöhnt sind und fügen Sie dann an allen Stellen, an denen später der Text aus der Tabelle stehen soll, eine Textmarke nach obigem Schema ein. Die Textmarken selbst sind leider unsichtbar.



Ein erster Entwurf der Vorlage für den Serienbrief der später über OLE-Automatisierung mit Daten gefüllt wird

Speichern Sie Ihr Word-Dokument in das Projektverzeichnis des Beispielprojekts *Tutorial*.

Sofern Sie dies noch nicht gemacht haben, fügen Sie bitte das Modul *OleWord.mod* (zu finden im Ordner *Lib* im Installationsverzeichnis von *TurboDB Studio*, also üblicherweise *C:\Programme\dataWeb\TurboDB Studio 4\Lib*) dem Ordner Kunden hinzu.

Öffnen Sie nun das Formularmodul *Modul_Kunden* des Formulars *Kunden* im Moduleditor: Klicken Sie zuerst mit dem + Symbol vor dem Formularsymbol. Klicken Sie dann das Modul *Modul_Kunden* mit der rechten Maustaste an und wählen Sie den Menüpunkt *Entwerfen* aus dem Kontextmenü.

Im Moduleditor sehen Sie bereits 2 Zeilen:

```
..formulargebundenes Modul für Formular Kunden
.SV 1
```

Schreiben Sie in die Zeile darunter folgende Anweisung:

```
uses OleWord
```

Mit dieser Anweisung wird die TurboPL-Funktions-Sammlung, die in *OleWord.mod* definiert ist, für das aktuelle Modul nutzbar gemacht.

Nun können wir mit dem eigentlichen TurboPL-Programm beginnen:

```
procedure DruckeEinladungen
  OleWordCreate(1)
```

```
  MoveBegin(Kunden)
  while ReadNext(Kunden)
    OleWordOpenDoc(BaseDir + 'Einladungsschreiben.doc')

    OleWordInsertText('Nachname', Kunden.Name)
    OleWordInsertText('Vorname', Kunden.Vorname)
    OleWordInsertText('Adresse', Kunden.Straße)
    OleWordInsertText('PLZ', Kunden.PLZ)
    OleWordInsertText('Ort', Kunden.Ort)
    OleWordInsertText('Datum', DateStr(Today))
    OleWordInsertText('Anrede', CHOICE(Kunden.Geschlecht, 'r Herr ', ' Frau ')
+ Kunden.Name)

    OleWordPrintDoc;
    OleWordCloseDoc(0);
  end;
  OleWordQuit;
endproc;
```

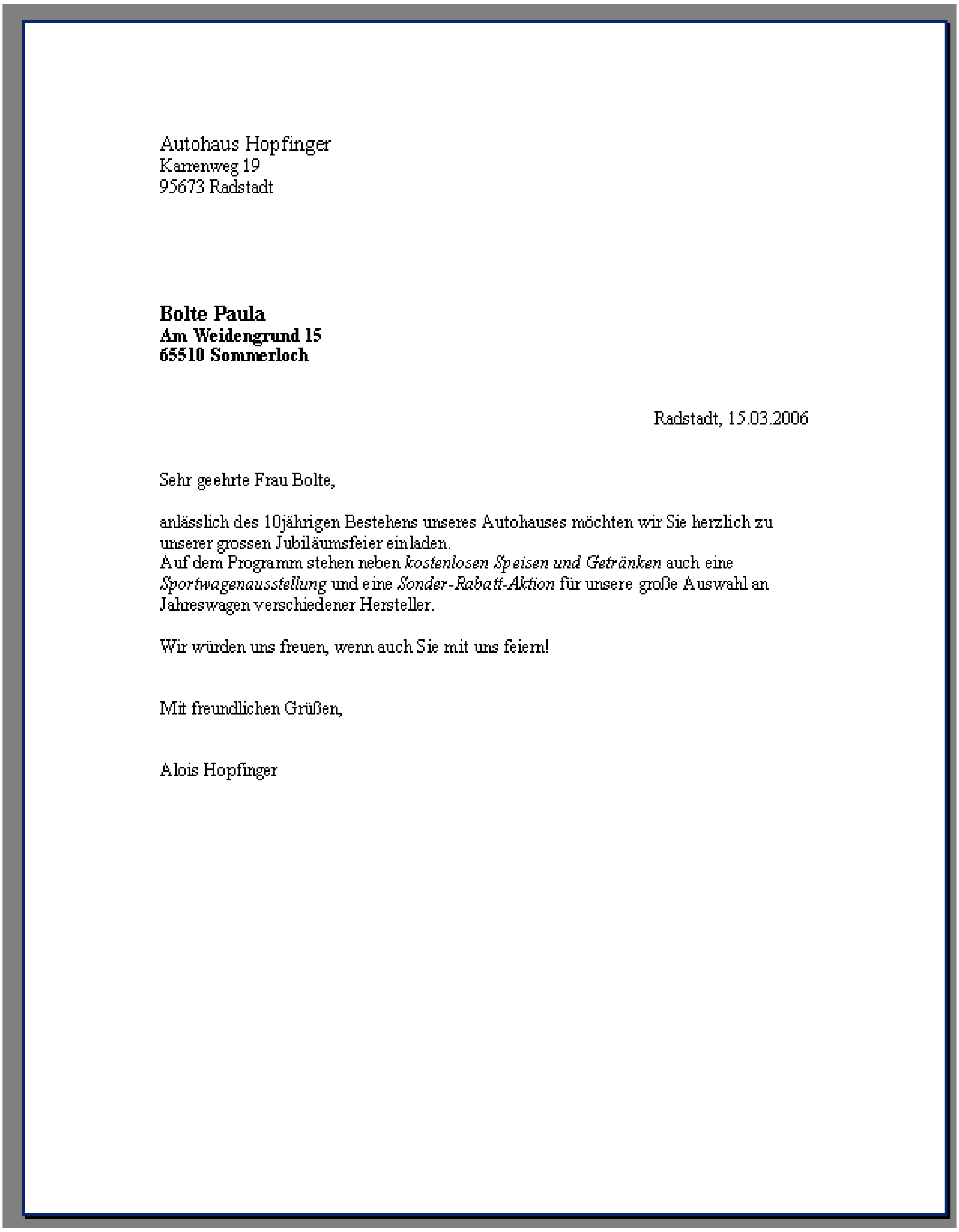
Diese Prozedur enthält nur die unbedingt benötigten Befehle um das Word-Dokument mit Daten zu füllen. Normalerweise sollten in einem guten Programm noch einige Dinge auf Ihre Richtigkeit geprüft werden, damit das Programm nicht bei jedem kleinen Fehler aus dem Tritt kommt. Beispielsweise sollte geprüft werden ob denn die Word-Datei, die als Vorlage dient, wirklich im Projektverzeichnis liegt und es sollte auch sichergestellt werden, dass trotz eines Fehlers das erzeugte, ferngesteuerte Word wieder beendet wird, da dieses sonst im Hauptspeicher bleibt bis Windows heruntergefahren wird (oder bis man den Windows Taskmanager zum Beenden des Prozesses benutzt). Da all diese Prüfungen den Programmcode jedoch unübersichtlicher machen, ist diese Version hier auf das wesentliche beschränkt.

Eine vollständige, mit Kommentaren versehene Version dieses Programms finden Sie im Modul *Modul_Kunden*. Um das vorgefertigte Programm auszuprobieren, müssen Sie lediglich die Kommentarzeichen *'..'* vor jeder Zeile entfernen. Markieren Sie dazu alle Zeilen die mit '..' beginnen und wählen Sie dann im Menü *Bearbeiten/Zeilen ent-kommentieren* oder drücken Sie *[Strg]+J*.

Das Programm startet Word, lädt automatisch das vorher erzeugte Word-Dokument und fügt Text an den Textmarken ein. Danach wird das Dokument gedruckt und wieder geschlossen. Dieser Vorgang wird für jeden Kunden in der Datenbank einmal ausgeführt bevor Word wieder beendet wird.

Wenn Sie vermeiden wollen, dass alle 36 Briefe gedruckt werden, entfernen Sie bitte vor dem Start des TurboPL-Programms das Papier aus Ihrem Drucker. Die von Word an Windows gesendeten Druckaufträge können Sie in der *Systemsteuerung* unter *Drucker* wieder löschen.

Mit dem Menübefehl *Modul/Prozedur ausführen...*, dem Schalter  in der Schalterleiste oder der Taste *[F9]* können Sie die OLE-Automatisierung testen. Je nachdem ob Sie die Funktion WordCreate() mit 1 oder 0 aufgerufen, können Sie Word bei der Arbeit zusehen (1) oder es bleibt unauffällig und unsichtbar im Hintergrund (0). In jedem Fall können Sie es (unter Windows 2000 und Windows XP) im Windows Taskmanager unter *Prozesse* als *WinWord* sehen.



Autohaus Hopfinger
Karrenweg 19
95673 Radstadt

Bolte Paula
Am Weidengrund 15
65510 Sommerloch

Radstadt, 15.03.2006

Sehr geehrte Frau Bolte,

anlässlich des 10jährigen Bestehens unseres Autohauses möchten wir Sie herzlich zu unserer grossen Jubiläumsfeier einladen.
Auf dem Programm stehen neben *kostenlosen Speisen und Getränken* auch eine *Sportwagenausstellung* und eine *Sonder-Rabatt-Aktion* für unsere große Auswahl an Jahreswagen verschiedener Hersteller.

Wir würden uns freuen, wenn auch Sie mit uns feiern!

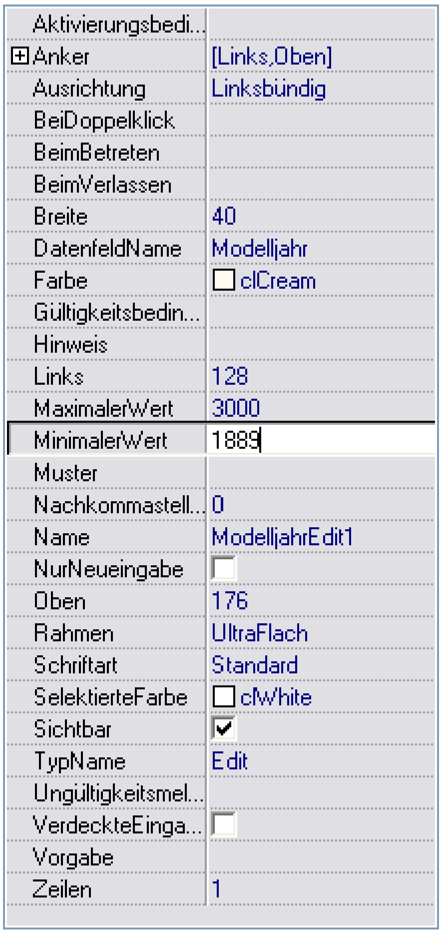
Mit freundlichen Grüßen,

Alois Hopfinger

Der mit Word und TurboDB Studio erstellte Serienbrief

2.14 Schritt 13: Die Dateneingabe kontrollieren

Ein wichtiger Punkt beim Entwurf von Formularen ist die Kontrolle der Dateneingabe. Dem Benutzer des Formulars soll es einerseits so einfach wie möglich gemacht werden, Daten zu erfassen und andererseits müssen die eingegebenen Daten so weit wie möglich auf Ihre Gültigkeit überprüft werden. Die Möglichkeiten von *TurboDB Studio* in diesem Bereich möchten wir Ihnen anhand des Formulars *Kraftfahrzeuge* der Tabelle *KFZ* veranschaulichen.



Die Eingabe in Datenfelder können Sie weitreichend regeln und überprüfen

Selektieren Sie dazu das Formular und öffnen Sie den Formulareditor für dieses Formular mit dem Knopf *Entwerfen*. Als erstes wählen Sie das Eingabefeld für das Modelljahr. Dadurch werden auf der linken Seite im Objektinpektor die Eigenschaften des Feldes angezeigt. Mit einem Blick auf die Eigenschaft *Minimalwert* sehen Sie, warum das Formular die Eingabe einer Jahreszahl wie 1860 ablehnt. Nur Werte zwischen 1889 und 3000 sind zugelassen. Wenn Sie diese Zahlen abändern, genügt es, das Formular mit Datei/Speichern zu sichern. Sie können den Formulareditor geöffnet lassen und die Veränderungen trotzdem sofort ausprobieren.

Für kompliziertere Überprüfungen haben Sie die Möglichkeit, eine beliebige Bedingung einzugeben. D.h. ein Kriterium wie bei der Suche oder dem Ausdruck von Listen. Nehmen Sie z.B. an, dass unter Bezeichnung nur Einträge mit mehr als drei Buchstaben erlaubt sein sollen. Dann tragen Sie für dieses Element bei der Eigenschaft Gültigkeitsbedingung folgenden Ausdruck ein:

```
Length(Bezeichnung) > 3
```

Bedingungen dieser Art sind im Kapitel Suchbedingungen ausführlicher beschrieben.

Um die Benutzerfreundlichkeit des Formulares zu erhöhen, sollten Sie bei einer fehlerhaften Eingabe dem Anwender mitteilen, was er denn falsch gemacht hat. Bei einer Bereichsüberprüfung erfolgt automatisch eine Meldung. Bei speziellen Bedingungen, die beliebig komplex und verschiedenster Art sein können, müssen Sie selber Hand anlegen. In unserem Fall könnten Sie zum Beispiel bei *Ungültigkeitsmeldung* eintragen:

```
Die Bezeichnung eines Fahrzeuges muss aus mehr als drei Zeichen bestehen.
```

Eine andere praktische Eigenschaft von Datenfeldern ist die Vorbelegung. Normalerweise sind die Einträge in neuen Datensätzen leer oder enthalten eine Null. Wenn Sie das anders haben wollen, tragen Sie unter *Vorgabe* einfach den gewünschten Wert ein. Bei der Anzahl Sitze auf der zweiten Seite des Formulars z.B. wäre 5 eine gute Vorgabe.

Der Vorgabewert lässt sich aber auch berechnen. Um bei Modelljahr die aktuelle Jahreszahl

vorzugeben, können Sie den Ausdruck

```
Year(today)
```

verwenden.

Manchmal hängt die Vorgabe in einem Feld von den Werten in anderen Feldern ab. Solche Fälle werden zwar durch die gewöhnlichen Vorgabewerte nicht abgedeckt, können aber z.B. mit dem Ereignis Beim Verlassen behandelt werden, das mit einem Makro verknüpft werden kann. Ein solches Makro enthält eine oder mehrere Anweisungen für *TurboDB Studio* in der integrierten Programmiersprache *TurboPL*, die immer dann ausgeführt werden, wenn ein Datenfeld geändert worden ist und auf ein anderes Feld umgeschaltet wird.

Nehmen wir als Beispiel die drei Felder *Brutto-Preis*, *Rabatt-Satz* und *Netto-Preis*. Sobald der Brutto-Preis eingetragen ist, erscheint der selbe Wert unter Netto-Preis als Vorgabe. Der Rabatt wird automatisch berechnet und ergibt sich natürlich zu 0%. Das Feld Netto-Preis ist aber editierbar, so dass das Auto auch für weniger verkauft werden kann. Das Rabatt-Feld passt sich automatisch jeder Änderung an. Wie funktioniert so etwas?

Zunächst muss die Vorgabe aus *Brutto-Preis* in *Netto-Preis* kopiert werden. Das Makro, das *BeimVerlassen* von *Brutto-Preis* ausgeführt wird, lautet

```
Netto-Preis := Brutto-Preis; Refresh;
```

Mit dem Doppelpunkt vor dem Gleichheitszeichen, weisen Sie daraufhin, dass der Wert von *Brutto-Preis* dem Feld *Netto-Preis* zugewiesen und nicht etwa mit ihm verglichen werden soll. Der anschließende Aufruf der Funktion *Refresh* sorgt dafür, dass die Änderung auch sichtbar wird. Der Strichpunkt trennt beiden Befehle voneinander. Jedesmal, wenn *Brutto-Preis* geändert wird, führt *TurboDB Studio* dieses Makro aus und kopiert somit die Eingabe in das Feld des Netto-Preises.

Damit auch noch der Rabatt korrekt berechnet wird, definieren Sie für *BeimVerlassen* von *Netto-Preis* den automatischen Eintrag

```
Rabatt-Satz := (Brutto-Preis - Netto-Preis) / Brutto-Preis * 100; Refresh;
```

Um auch noch bei einer Änderung von *Rabatt-Satz* den Netto-Preis anzupassen, definieren wir zu Schluss noch ein Makro, das *BeimVerlassen* von Rabatt-Satz ausgeführt wird:

```
Netto-Preis := Brutto-Preis - ((Brutto-Preis / 100) * Rabatt-Satz); Refresh;
```

Beachten Sie den Unterschied zwischen einem Bindestrich und einem Minus-Zeichen. Als deutsches Datenbankprogramm erlaubt *TurboDB Studio* die Verwendung von Bindestrichen in Bezeichnern wie in *Brutto-Preis* und in *Netto-Preis*. Damit keine Verwechslung mit dem Minus-Zeichen möglich ist, sollte vor und hinter dem Minus-Zeichen ein Leerzeichen stehen. Falls nicht, kann das zur Fehlermeldung *Unbekannter Bezeichner* führen, weil *TurboDB Studio* dann versucht, ein Feld namens *Brutto-Preis-Netto-Preis* zu suchen.

2.15 Schritt 14: User-Modus

Einer der Hauptgründe, warum Datenbankprogramme immer als besonders kompliziert angesehen werden, liegt darin, dass sogar das Arbeiten mit einem schon fertiggestellten Projekt immer ein gewisses Verständnis für Datenbanken voraussetzt. Im allgemeinen braucht man also einen Anwendungsprogrammierer, der eine Oberfläche für die all die Tabellen, Formulare und Berichte erstellt, die zu einem Projekt gehören. Erst dann kann man die Datenbank im Büro effizient einsetzen.

Aus diesem Dilemma befreit Sie *TurboDB Studio* weitgehend. Wenn Sie einmal ein Projekt zusammengestellt haben, erzeugt der *TurboDB Studio Anwendungs-Generator* selbständig und ohne einen weiteren Handgriff ein lauffähiges Programm daraus, so dass Sie Ihr Projekt als eine eigenständige Windows-Anwendung auch an unerfahrene Benutzer weitergeben können.

Um das zu demonstrieren, öffnen Sie einmal ihr Projekt und wählen dann *Datei/Anwendung testen* . Das Projekt wird geschlossen und dann im User-Modus wieder geöffnet. Was Sie nun sehen, ist Ihre Datenbank-Anwendung in der nur noch Datenbearbeitung aber kein Entwurf mehr möglich ist.

Diese Anwendung können Sie auch als eigenständiges Programm erzeugen. Beenden Sie den *User-Modus* mit *Datei/Beenden*. Dadurch gelangen Sie in den *Autoren-Modus* zurück. Nun starten Sie mit *Datei/Anwendung erzeugen* den *Anwendungs-Generator*. Dieser erzeugt in wenigen Sekunden eine EXE-Datei im Verzeichnis ihres Projektes.

Damit ist Ihre Datenbankanwendung fertig. Mit einem Doppelklick auf das neu erstellte Programm

starten Sie die Applikation KFZ im *User-Modus*. In dieser Form können Sie Ihr Projekt getrost von anderen benutzen lassen. Jeder, der ein bisschen Ahnung von Windows-Programmen hat, kann mit dieser Applikation umgehen.

> Die EXE-Datei dürfen Sie nur dann an andere weitergeben, wenn Sie registrierter Besitzer der Professional- oder Workgroup-Edition sind. Die mit der Standard-Edition erzeugten EXE-Dateien sind ausschließlich für die Benutzung durch Sie selbst gedacht!

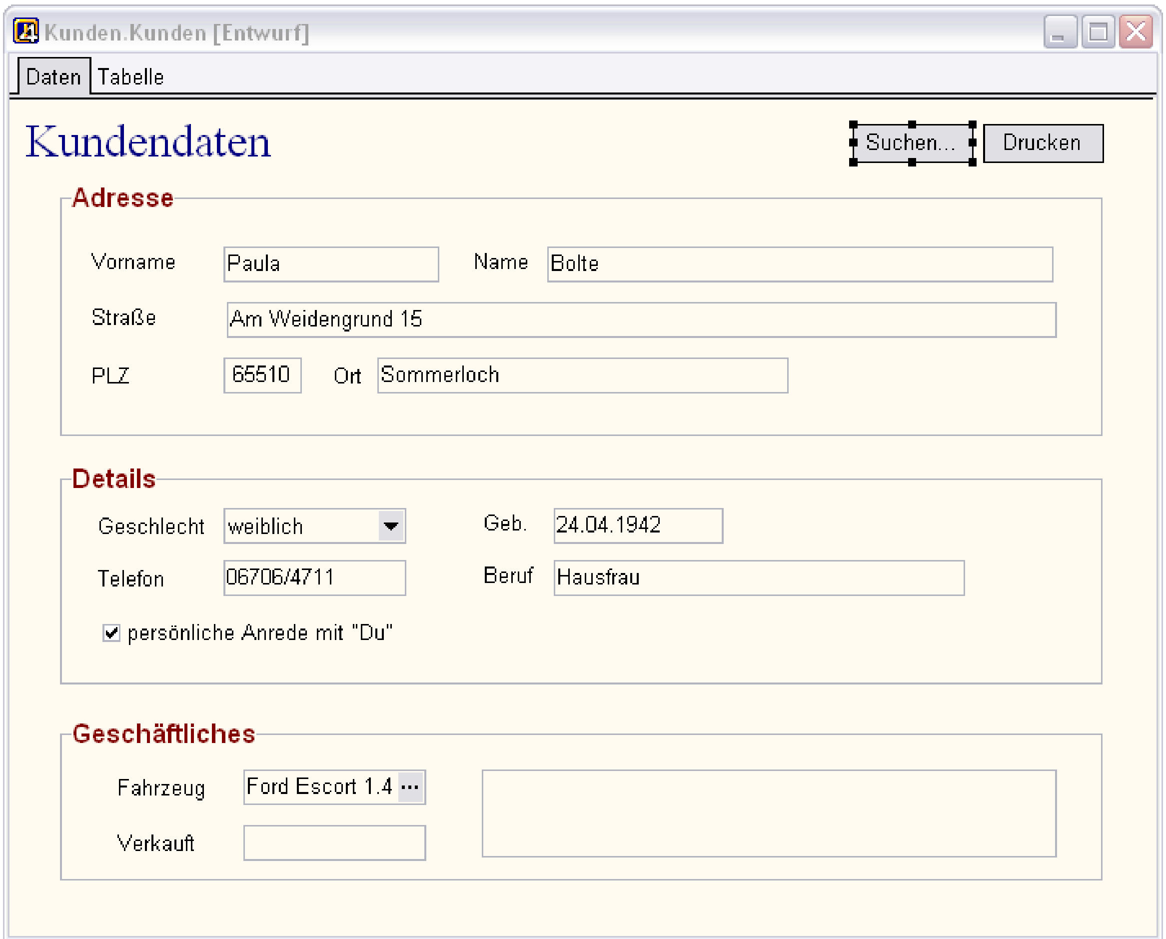
Obwohl Sie keine Zeile programmieren müssen, um eine Windows-Anwendung für sich oder andere zu erstellen, können Sie viele Eigenschaften selbst festlegen. Durch einfaches Erstellen von Formularen, Verknüpfen von Elementen und Ändern von Eigenschaften bestimmen Sie die Oberfläche Ihrer Applikation. Sie können sogar ein eigenes Start-Logo integrieren, einfach, indem Sie eine beliebige BMP-Datei unter dem Namen LOGO.BMP in das Verzeichnis des Projektes kopieren. Lesen Sie das Kapitel *Anwendungen erstellen*, wenn Sie die Feinheiten dieser Form der Applikationserstellung kennenlernen wollen.

2.16 Schritt 15: Makros

Je weiter eine Anwendung fortschreitet, desto mehr Features und Annehmlichkeiten möchte man dem Benutzer bieten. Obwohl Ihre *TurboDB Studio Applikation* schon von Beginn an alles hat, was man zum Pflegen und Auswerten der Daten benötigt, kommt man irgendwann bei den Zusatzfunktionen und maßgeschneiderten Eingabehilfen einmal an den Punkt, wo man wirklich selbst Hand anlegen muss. Mit Hilfe von Makros, selbstdefinierten Schaltern und schließlich auch eigenen Menüpunkten können Sie fast alle Ihre Wünsche erfüllen.

Als kleines Beispiel versehen wir das Kunden-Formular mit einigen Knöpfen zur vereinfachten Bedienung. Einer soll die Suche nach einem bestimmten Kunden vereinfachen, der andere den Serienbrief ausdrucken.

1. Entwerfen Sie das Kundenformular.
2. In der Palette sehen Sie auf der Seite der *Standard-Elemente* ein Symbol mit der Beschriftung *Schalter*. Ziehen Sie zwei davon auf Ihr Formular.
3. Bei den Eigenschaften des ersten Schalters tragen Sie bei der Eigenschaft *Text* den Text *Suchen...* ein.
4. Weiter oben finden Sie die Eigenschaft *BeimAnklicken*, das die Befehle aufnehmen soll, die beim Anklicken ausgeführt werden. Hier tragen Sie folgendes ein:
   ```
   Run("KUNDEN.Rundschreiben", 2)
   ```
5. Beim zweiten Schalter schreiben Sie *Suchen...* in das Feld der Eigenschaft *Text*.
6. Da dieser Schalter mehr als einen Befehl enthält, wäre es recht unübersichtlich, wenn man das alles in die kurze Zeile quetschen würde. Doppelklicken Sie deshalb die Eigenschaft *BeimAnklicken* oder wählen Sie aus dem Kombinationsfeld den Eintrag *<neues Makro>*. Es öffnet sich der Texteditor und TurboDB Studio legt im Modul auch gleich einen sogenannten Funktionsrumpf für Sie an: Das Gerüst eines Makros, das Sie nur noch mit Inhalt füllen müssen.
7. Schreiben Sie nun folgende Befehle zwischen die Zeilen *procedure* und *endproc*:
   ```
     VarDef name: String;
     Input('Name des gesuchten Kunden:', '', 0, name);
     Suchen("Kunden.id", name);
   ```
8. Speichern Sie das Modul und schließen Sie den Editor.
9. Wenn Sie wollen, können Sie noch den *Rahmen* der Schalter anpassen, damit sie optisch zum Rest des Formulars: *UltraFlach*.
10. Speichern Sie Ihre Änderungen am Formular und testen Sie es dann durch einen Klick auf den *Formular Testen*-Schalter oder durch die *[F9]*-Taste.
11. Der Schalter *Drucken* öffnet die Druckvorschau und zeigt Ihnen das Rundschreiben an die Kunden.
12. Versuchen Sie nun den Schalter *Suchen...*. Ein Dialogfeld mit Eingabefeld fordert Sie auf, einen Namen einzutragen. Tippen Sie *Renn* ein und drücken Sie dann *Ok*. Im Formular wird ein passender Eintrag gesucht und "Eddi Rennfahrer" gefunden. Falls Sie stattdessen eine Fehlermeldung erhalten, vergleichen Sie die obigen Angaben noch einmal mit Ihrem Formular.



Mit Schalter können Sie selbst erstellte TurboPL-Makros auslösen und so dem Benutzer die Arbeit erleichtern

Was steckt dahinter? Hier der Blick hinter die Kulissen:

Beim ersten Schalter genügt ein einziger Aufruf, um die gewünschte Aktion auszulösen. Der TurboPL-Befehl *Run* führt einen Bericht oder Datenbankjob aus. Welchen, das bestimmen Sie indem Sie den vollständigen Namen des Berichts bzw. Datenbankjobs in Klammern hinter den Befehl schreiben. Alles was in Klammern hinter einem TurboPL-Befehl steht, wird als Parameter (auch als Argument genannt) übergeben und bestimmt, was genau der Befehl machen soll. Der zweite Paramter des Befehls bestimmt, wohin der Ausdruck erfolgen soll. Mit 0 wird der Druck-Dialog aufgerufen, 1 druckt den Bericht ohne Nachfrage auf dem in Windows eingestellten Standard-Drucker, 2 startet ohne Nachfrage die Druckvorschau und 3 druckt gar nicht, sondern leitet die Ausgabe in eine Datei um, deren Format Sie vorher mit mit dem Befehl *SetzeAusgabeDatei* festlegen können.

Beim zweiten Schalter passiert schon etwas mehr. Die verwendeten Befehle sollen nun etwas ausführlicher erklärt werden.

VarDef teilt TurboDB Studio mit, dass es eine Variable anlegen soll. V*arDef name* bedeutet folglich, dass die angelegte Variable unter dem Bezeichner *name* zur Verfügung stehen soll. V *arDef name: String* teilt *TurboDB Studio* mit, dass in der neuen Variable eine Buchstabenkette (engl.: String = Zeichenkette) - also Text - gespeichert werden soll.
Input öffnet ein Fenster mit einem Text und einem Eingabefeld. In dieses trägt der Benutzer etwas ein und drückt dann *Ok* oder *Abbruch*. Was er eingetippt hat steht anschließend in unserer Variablen *name* und kann weiterverarbeitet werden. Das Dialogfeld von *Input* hat normalerweise keine Überschrift, Sie können jedoch eine festlegen, indem Sie hinter dem ersten Text noch einen zweiten angeben:

```
Input("Name des gesuchten Kunden:", "Nach einem Kunden suchen", 0, name);
```

Die *0* nach dem Text für die Überschrift bedeutet, dass es sich bei dem eingetippten Text nicht um

ein Passwort handelt. Schreibt man an diese Stelle eine 1, so wird der eingegebene Text als Sternchen dargestellt, so wie das bei Passworteingaben unter Windows üblich ist.

Die Prozedur (anderes Wort für Funktion, Befehl) *Suchen* entspricht dem Menüpunkt *Suchen*. Eine Index-Suche nach dem angegebenen Text wird durchgeführt. Der zu verwendende Index steht im ersten Argument des Aufrufs, der Suchbegriff im zweiten. In unserem Beispiel wird nach einem Eintrag gesucht, bei dem der Name mit einer bestimmten Zeichenkombination beginnt. Die Indexdatei *Kunden.id* enthält einen Index über *Name* und *Vorname* und kann also gut verwendet werden. Der Suchbegriff selbst wurde von der Funktion *Input* in die Variable *name* eingetragen und kann direkt an die Funktion *Suchen* weitergegeben werden.

Sie sehen, auch die Arbeit mit Makros ist mit *TurboDB Studio* sehr einfach. Die Makrosprache umfaßt einige hundert verschiedene Befehle von A wie *ActivateForm* (öffnet ein Formular) bis Z wie *Zurückblättern* (zeigt die vorherige Seite im Formular an). Eine vollständige Liste nach Sachgebieten geordnet finden Sie in der Hilfe unter *TurboPL Referenz*.

Mehr über TurboPL und das Schreiben von Makros erfahren Sie in der Hilfe im Kapitel *Aufgaben / Makros und Programme einsetzen*.

2.17 Schritt 16: Volltextsuche

Neben der normalen Suche mit Suchbedingung und der direkten Suche über Index bietet *TurboDB Studio* als dritte Suchoption die Stichwortsuche an. Im Grunde ist das die einfachste Art, weil lediglich ein oder mehrere Begriffe einzugeben sind. Gefunden werden alle Datensätze, die irgendwo diese Begriffe enthalten.

Die Mächtigkeit einer Volltextsuche zeigt sich in erster Linie bei Tabellen, die überdurchschnittlich viel Text in Form alphanumerischer Felder und Memos beinhalten. Als Beispiel soll hier das Beispielprojekt *Vögel* dienen, das im Auslieferungsumfang enthalten ist. Der normale Weg, um eine Stichwortsuche auf allen Datensätzen durchführen zu können, wäre der folgende: Man legt eine eigene Tabellenspalte mit den Schlagworten an, auf denen man dann eine Suche starten kann. Nachteil dieser Methode ist jedoch die Redundanz (also das überflüssige Vorkommen) vieler Wörter und der dadurch erhöhte Speicherplatzbedarf. Sinnvoller wäre es eine eigene Tabelle von Stichworten anzulegen, und in der Tabelle mit den Vögeln lediglich ein Relationsfeld mit Verweisen auf diese Worte vorzusehen. Dies erfordert natürlich einen gesteigerten Arbeitsaufwand bei der Eingabe der Daten des jeweiligen Vogels.

Der *Volltext-Index* bietet Ihnen nun diese bessere Alternative und nimmt Ihnen den zusätzlich entstehenden Arbeitsaufwand ab. Beachten Sie, dass das Anlegen eines solchen Indexes die Voraussetzung für die Benutzung der Volltextsuche ist. Seine Erstellung gestaltet sich aber wie gewohnt einfach, da Sie hier wieder von einem benutzerfreundlichen Assistenten unterstützt werden.

Um nun eine Volltextsuche in der Vögel-Datenbank durchführen zu können, müssen wir zunächst einen Volltext-Index auf der Tabelle VÖGEL generieren. Zu diesem Zweck schließen Sie zunächst alle Fenster der Tabelle und öffnen dann das Datenmodell mit einem Doppelklick auf das Symbol im Projektnavigator oder mit dem Schalter *Ausführen*. Im Datenmodell sehen Sie für jede Tabelle ein kleines Fenster mit der Tabellenstruktur. Selektieren Sie hier die Tabelle *VÖGEL* und klicken Sie mit der rechten Maustaste auf das Fenster. Wählen Sie im Kontextmenü *Volltext-Index erstellen*. Alternativ können Sie auch im Menü *Tabelle/Volltextindex erzeugen...* anklicken.

Nun meldet sich der Volltext-Assistent und fordert Sie zur Eingabe eines Namens für die zu erstellende Stichworttabelle auf. Ein treffender Name wäre zum Beispiel *Stichworttabelle*. Wie schon oben erwähnt, wird in die Ausgangstabelle - also *VÖGEL* - ein Relationsfeld eingefügt. Für dieses Feld schlägt der Assistent den Titel *Stichwörter* vor. Sollte Ihnen dieser Name nicht zusagen, können Sie ihn gleich hier ändern, ansonsten können wir mit Weiter zu neuen Taten schreiten. Beachten Sie bitte, dass der Name des Feldes für die Stichwörter und der Name der Stichworttabelle nicht gleich sein dürfen.

Die nächste Seite enthält zwei Auswahlboxen. Die linke enthält alle alphanumerischen Felder und Memos - die potenziellen Bestandteile des Volltext-Indexes. Mit den Knöpfen  /  und  /  können Sie, wie von anderen Dialogen schon bekannt, nach Belieben diejenigen Felder in die rechte Liste eintragen (bzw. entfernen), deren Inhalte anschließend die Grundlage des Indexes bilden. Wir wollen hier einmal alle Spalten aufnehmen.

Worte, die für die Volltextsuche nicht von Bedeutung sind, führen zu einer unnötigen Aufblähung der Stichworttabelle und sind beim Suchvorgang nur hinderlich. Solche Worte kommen meist in großer Zahl vor. Aus diesem Grund sollten sie durch die Angabe einer Obergrenze herausgefiltert werden. Diesem Filterprozess widmet sich die folgende Seite des Assistenten. Tritt ein Wort zum Beispiel mehr als eintausend mal auf, so ist die Wahrscheinlichkeit, dass es ein spezielles Charakteristikum eines Datensatzes darstellt, ziemlich gering. Deshalb wird es dann aus der Stichworttabelle entfernt und alle Verweise darauf werden gelöscht. Natürlich hängt die Obergrenze von der Größe der Tabelle ab. Enthält eine Tabelle 1 Million Datensätze, so kann der Wert 1000 auch zu gering ausfallen und auch charakteristische Schlagworte würden der Stichworttabelle vorenthalten. Füllwörter wie *und*, *oder*, *weil*, etc. besitzen äußerst selten einen Informationsgehalt. Sie sollten von vornherein von einer Aufnahme in die Stichworttabelle ausgeschlossen werden. Zu diesem Zweck sollte man eine Datei anlegen, die solche Worte - jeweils durch einen Zeilenumbruch getrennt - enthält. So eine Datei kann mit jedem Texteditor oder Textverarbeitungsprogramm erstellt werden. Dabei muss nur darauf geachtet werden, dass sie im *.txt-Format abgespeichert wird. Durch einen Druck auf den Schalter [...] erscheint ein Datei-Auswahldialog, mit dem Sie die gewünschte Textdatei spezifizieren können.

Jetzt wird es ernst: wenn Sie auf der letzten Seite des Pfadfinders mit *Ausführen* bestätigen, so können Sie in der Anzeige den Fortschritt der Volltextindex-Erstellung beobachten. Je nach Größe der Tabelle kann dies auch mehrere Minuten in Anspruch nehmen, bei unserer *VÖGEL*-Tabelle ist dies aber nicht der Fall.

Nun können wir die Vorteile der Volltextsuche nutzen. Dazu wechselt man zuerst in ein Datenfenster der Tabelle - also entweder ein Formular oder das Tabellenfenster - und wählt im Menü Suchen/Volltext... . Im erscheinenden Dialog muss man nun die erforderliche Stichworttabelle mit Hilfe der Auswahlbox bestimmen. Jetzt kann die Suche beginnen. Geben Sie einfach in die folgenden Eingabefelder bis zu drei Suchbegriffe ein oder nutzen Sie die erweiterten Möglichkeiten, indem Sie die Suchbedingung direkt eingeben. Die Verwendung der Operatoren entnehmen Sie bitte der Beschreibung im Abschnitt *Suchbedingungen*.

2.18 Schritt 17: Datenbankjobs erstellen

In manchen Fällen kommt es darauf an, viel Text möglichst einfach mit einigen Datenfeldern zu mischen. Die Datenbankjobs von *TurboDB Studio* sind für diese Aufgabe ideal.

Wenn Sie auf die bekannte Art und Weise einen neuen Datenbankjob zum Ordner *Kunden* hinzugefügt haben, befinden Sie sich im Texteditor, um den Text für Ihren Serienbrief einzugeben. Der könnte z.B. so aussehen:

```
Autohaus Hopfinger
Karrenweg. 19
95673 Radstadt


$Vorname $Name
$Straße
$PLZ $Ort


                                             Radstadt, den $heute



Sehr geehrter Herr $Name,

nach unseren Unterlagen haben Sie kürzlich einen $Fahrzeug.Bezeichnung der
Firma $Fahrzeug.Hersteller bei uns erworben.
Leider wurde für dieses Fahrzeug ein sicherheitskritischer Produktionsfehler
bekannt, der schnellstmöglich repariert werden muß.
Wenden Sie sich deshalb bitte umgehend an unsere Zentrale in Radstadt.
```

```
Mit freundlichen Grüßen


Alois Hopfinger
```

Wie Sie sehen, besteht ein solcher Datenbankjob aus ganz normalem Text, in den die gewünschten Datenfelder eingefügt und mit $ gekennzeichnet werden. Wie das Ergebnis aussieht, können Sie sofort mit *Datenbankjob/Vorschau* bzw. *[F9]* kontrollieren.

Leider hat dieser Brief einige Unzulänglichkeiten. Der auffälligste Fehler ist, dass auch weibliche Kunden wie beispielsweise Paula Bolte mit "Herr" angesprochen werden. Das darf natürlich nicht sein. Um Probleme wie dieses zu lösen, gibt es spezielle Kommandos für Datenbankjobs, die sich durch einen Punkt in der ersten Spalte von normalem Text unterscheiden. Ersetzen Sie Zeile

```
Sehr geehrter Herr $Name,
```

durch die folgenden Zeilen:

```
.FALLS $Geschlecht = weiblich
Sehr geehrte Frau $Name,
.SONST
Sehr geehrter Herr $Name,
.ENDE
```

Wenn Sie es bevorzugen, können Sie anstatt der deutschen Begriffe *FALLS*, *SONST* und *ENDE* natürlich auch deren englische Pendants *IF*, *ELSE* und *END* benutzen.

Wahrscheinlich sehen Sie schon, was passiert. Durch das Kommando *FALLS* wird je nach dem Wert des Datenfeldes *Geschlecht* eine der beiden Anredezeilen ausgesucht und gedruckt. In der Druckvorschau erkennen Sie, dass alle Kunden jetzt mit der ihnen zustehenden Anrede angesprochen werden.

Prinzipiell gilt:
Eine Zeile, die mit einem Punkt beginnt enthält Befehle und wird nicht gedruckt. Die Befehle können entweder Kommandos wie *FALLS*, *SONST* und *ENDE* sein oder Steuerbefehle für den Drucker. Zu letzteren gehören beispielsweise *PO* und *MT*, mit deren Hilfe Sie den linken und oberen Seitenrand (Page Offset und Margin Top) einstellen können. Wenn Sie also einen Rand von acht Spalten links und vier Zeilen oben haben möchten, tragen Sie zu Beginn des Serienbriefs die Zeile

```
.PO 8, MT 4
```

ein.

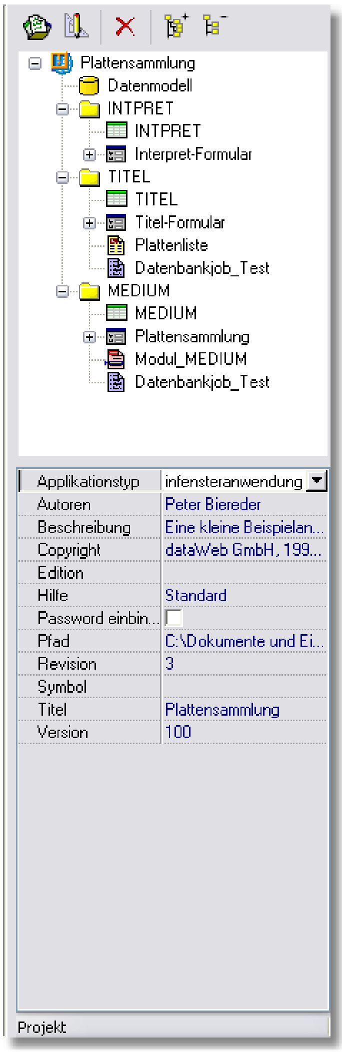
Eine Beschreibung aller Möglichkeiten im Serienbrief bringt das Kapitel *Datenbankjobs erstellen*.

3 Aufgaben

3.1 Projekte verwalten

Ein Projekt besteht aus einer Datenbank, den Formularen zur Dateneingabe, Datenbankjobs und Berichten zum Drucken von Auswertungen sowie Modulen mit dem TurboPL-Quellcode für Makros. Alle diese Element werden im Projektfenster dargestellt und können von hier aus verwaltet, geöffnet und ausgeführt werden.

Wenn Sie im Projektfenster ein Element selektieren, zeigt das Eigenschaftsfenster darunter dessen Einstellungen an:



Im Projektfenster hat jedes Element einen Titel und einen Dateinamen. Bei allen Elementen außer Tabellen können diese beiden Eigenschaftn völlig verschieden sein. Ein Formular kann zum Beispiel den Dateinamen C:\Meine Projekte\P1\FormularA.frm haben aber im Projekt als Neueingabe Kunde angezeigt werden. Es wird jedoch der besseren Übersicht halber empfohlen, den Dateiname und den Titel gleich zu wählen, d.h. das Formular als C:\Meine Projekte\P1\Neueingabe Kunde.frm abzuspeichern.

3.1.1 Ein neues Projekt anlegen

So legen Sie ein neues Projekt an:

1. Wählen Sie den Befehl *Datei/Neu/Projekt...* aus dem Menü. Der Dateidialog zur Auswahl der Datenbank wird geöffnet:
2. Sie können nun entweder eine schon vorhandene Datenbank auswählen oder eine neue Datenbank für das Projekt erzeugen. Datenbanken können entweder aus einzelnen Dateien für alle Tabellen und Indexe bestehen (Verzeichnis-Datenbank) oder aus einer einzigen Datei für alle Datenbankobjekte (Single-File Datenbank).
3. Bestimmen Sie einen Ordner und einen Namen für das Projekt und bestätigen Sie.
4. Das Projekt wird nun erstellt. Im nächsten Schritt sollten Sie die Datenbanktabellen definieren, indem Sie Tabellen zum Projekt hinzufügen.

3.1.2 Projekt-Einstellungen vornehmen

Unter dem Menüpunkt *Datei/Einstellungen* können Sie generelle Einstellungen für das geöffnete TurboDB Studio-Projekt vornehmen:



- *Installationsverzeichnis*: In diesem Verzeichnis werden Tabellen gesucht, deren Eigenschaft Suchpfad auf Resident eingestellt wurden.
- *Nachfrage bei der Eingabe angekoppelter Datensätze*: Wenn diese Option markiert ist, zeigt TurboDB Studio bei der Eingabe von Werten in ein Koppel- oder Relationsfeld auch dann die Nachschlagetabelle an, wenn der eingetippte Wert erlaubt ist, d.h. in der Nachschlagetabelle schon existiert.
- *Name des Rechners*: Unter diesem Namen werden Zugriff auf Datenbanken und Tabellen verwaltet und erscheinen auch im Netzmonitor. Dieser Name muss nicht unbedingt mit dem Windows-Namen des Rechners übereinstimmen und es schadet auch nichts, wenn er leer bleibt. Allerdings ist er oft von Nutzen, wenn man beispielsweise wissen möchte, wer noch auf eine bestimmte Tabelle zugreift.
- *Privates Verzeichnis*: In diesem Verzeichnis werden temporäre private Daten abgelegt, wie zum Beispiel die Ergebnisse von SQL-Abfrage und manche temporäre Indexe. Wenn dieser Eintrag leer ist, wird das Standard-mäßige Verzeichnis für temporäre Dateien benutzt.
- *Logging*: TurboDB Studio kann Fehler beim Zugriff auf Tabellen in eine Datei loggen, die im privaten Verzeichnis angelegt wird.

3.1.3 Eigenschaften des Projekts einstellen

Die Eigenschaften des Projekts bestimmen hauptsächlich die Eigenschaften der erzeugten Anwendung. Sie werden im Eigenschaftsfenster angezeigt, wenn der oberste Knoten im Projektfenster selektiert ist.

- *Applikationstyp*: Definiert das Fenstermodell der Anwendung:
  - *Einfensteranwendung*: Es ist immer nur ein Formular zur Zeit sichtbar.
  - *Mehrfensteranwendung*: Der Anwender kann mehrere Fenster gleichzeitig öffnen und zwischen ihnen wechseln.

- *Autoren*: Eintrag für die Namen der Anwendungsentwickler.
- *Beschreibung*: Eintrag für eine Beschreibung der Anwendung.
- *Copyright*: Eintrag für das Copyright der Anwendungsentwickler.
- *Edition*: Eintrag für die Edition der Anwendung.
- *Hilfe*: Art der Online-Hilfe-Unterstützung.
- *Passwort einbinden*: Legt fest, ob das Passwort mit in die Anwendung aufgenommen wird. Diese Option beseitigt für den Anwender die Notwendigkeit, das Passwort beim Start der Anwendung einzugeben. Andererseits bedeutet das auch, dass jederman mit Zugang zur Anwendung diese starten kann.
- *Revision*: Eintrag für die Revisionsnummer der Anwendung.
- *Symbol*: Pfad zur Ikone der Anwendung.
- *Titel*: Titel der Anwendung. Wird im User-Modus als Überschrift im Hauptfenster der Applikation angezeigt.
- *Version*: Versionsnummer der Anwendung. 100 ist die Version 1.0, 150 die Version 1.5 etc.

3.1.4 Neue Projektelemente erstellen

Wenn Sie ein Projekt aufbauen oder erweitern, müssen Sie neue Tabellen, Formulare, Berichte, Datenbankjobs und Module erstellen. Die zugehörigen Befehle finden Sie unter *Datei/Neu* oder im Kontext-Menü des Projektfensters.

In TurboDB Studio werden Formulare, Berichte, Datenbankjobs und Module immer einer einer Tabelle, der Primärtabelle zugeordnet. Diese Zuordnung vereinfacht die Angabe von Spaltennamen. Z.B. ist es dadurch klar, zu welcher Spalte eine Feld *Name* gehört, auch wenn in der Datenbank mehrere Tabellen mit einem Feld *Name* vorhanden sind.

So erstellen Sie eine neue Tabelle für das aktuelle Projekt:

1. Rechtsklicken Sie im Projektfenster und wählen Neu/Tabelle...
2. Tragen Sie im Eingabefeld einen Namen für die neue Tabelle ein und bestätigen Sie.
3. Im Tabellen-Designer definieren Sie die Eigenschaften und Spalten der neuen Tabelle und klicken auf OK.
4. Im Projektfenster wird ein neuer Ordner für die Tabelle angezeigt. Die eigentliche Tabelle ist in diesem Ordner zu finden.
5. Wenn Sie die Tabelle im Projektfenster auswählen, können Sie darunter deren Eigenschaften sehen und abändern.

So erstellen Sie ein neues Formular, einen neuen Bericht, einen neuen Datenbankjob oder ein neues Modul für das aktuelle Projekt:

1. Markieren Sie den Ordner, in dem Sie das Projektelement erstellen wollen im Projektfenster.
2. Wählen Sie im Kontextmenü den Eintrag Neu und dann den Typ des Elements.
3. Legen Sie den Namen und ggf. den Speicherort des neuen Projektelements fest und bestätigen Sie.
4. Das neue Element wird im Projektfenster eingetragen und ein Designer zum Bearbeiten öffnet sich.

3.1.5 Projektelemente hinzufügen

Sie können bestehende Projektelemente zu einem Projekt hinzufügen. Das ist vor allem dann sinnvoll, wenn das selbe Element in mehreren Projekten verwendet wird oder wenn Sie bestehende Tabellen der *Turbo Datenbank für DOS* oder von *Visual Data Publisher* in ein *TurboDB Studio*-Projekt aufnehmen wollen.

So fügen Sie ein bestehendes Projektelement zum aktuellen Projekt hinzu:

1. Markieren Sie die Tabelle im Projektfenster, zu der Sie das Element hinzufügen wollen. Wenn Sie ein Tabelle hinzufügen wollen, entfällt dieser Punkt natürlich.
2. Wählen Sie den Befehl Hinzufügen im lokalen Menü und geben Sie im Dialogfenster die Datei

des gewünschten Projektelements an.

3. Bestätigen Sie Ihre Eingabe mit OK und das ausgewählte Element wird dem Projekt hinzugefügt.

Bei Tabelle ist es insofern etwas anderes, als Tabellen aus Datenbanken-Datein natürlich nicht zu einem anderen Projekt hinzugefügt werden können. Sie können allerdings eine einzelne Tabelle (eine dat-Datei) einem Projekt hinzufügen, das auf einer Single-File-Datenbank basiert.

Hier ein kleines Anwendungsbeispiel:

Nehmen Sie an, Ihre Datenbank besteht aus den Tabellen für Kundenstamm (KUNDEN), Aufträge (AUFTRAG) und Rechnungen (RECHNUNG). Dann könnten Sie für die Pflege des Kundenstammes, die Auftragsverwaltung und die Rechnungserstellung drei verschiedene Projekte anlegen. Dies ist vor allem dann sinnvoll, wenn unterschiedliche Mitarbeiter mit diesen Aufgaben betraut sind.

Das Projekt KUNDEN würde dann nur die Tabelle KUNDEN mit einem Formular für die Eingabe und einen Bericht zum Drucken von Adressaufklebern umfassen.

Im Projekt AUFTRAG, wären sowohl die Kunden-Tabelle als auch die Auftrags-Tabelle enthalten. Außerdem ein Formular zum Erfassen der Aufträge und ein Bericht zum Ausdrucken aller noch offenen Aufträge.

Für die Rechnungserstellung schließlich nehmen Sie alle drei Tabellen in das Projekt RECHNUNG auf und fügen ein Formular für die Eingabe hinzu sowie zwei Berichte zum Drucken von Rechnungen und Mahnungen. Evtl. ist auch das Adressetikett aus dem KUNDEN-Projekt nützlich.

3.1.6 Projektelemente bearbeiten

Zum Bearbeiten der Projektelemente im Projektfenster gehört auch das Verschieben und Entfernen.

So verschieben Sie ein Projektelement inerhalb des Projektes:

1. Klicken Sie das Element mit der linken Maustaste.
2. Ziehen Sie es an die gewünschte Stelle.

Bitte beachten Sie:

Wenn Sie eine Element von einer Tabelle zu einer anderen ziehen, kann das dazu führen, dass es sich anders verhält. Sollte sowohl die vorherige als auch die neue Tabelle eine Spalten namen Column1 enthalten, wird sich ein solcher Verweis im Modul oder Datenbankjob jetzt auf die neue Tabelle beziehen.

So entfernen Sie ein Projektelement aus dem Projekt:

1. Selektieren Sie das Projektelement im Projektfenster.
2. Wählen Sie im Kontext-Menü den Befehl Entfernen.

Bitte beachten Sie:

Die Dateien von Projektelemente werden in TurboDB Studio grundsätzlich nicht gelöscht. Sie könnten ja noch für andere Projekte verwendet werden. Wenn Sie ein Element endgültig löschen wollen, entfernen Sie es zuerst aus dem Projekt und löschen dann die zugehörige Datei auf der Festplatte mit dem Windows Explorer.

So entfernen Sie eine Tabelle aus dem Projekt:

1. Selektieren Sie die Tabelle im Projektfenster.
2. Wählen Sie im Kontext-Menü den Befehl *Entfernen*.

Bitte beachten Sie:

Beim Entfernen der Tabelle werden auch alle Projektelemente entfernt, die zu dieser Tabelle gehören. Sie sind aber nicht gelöscht und können jederzeit wieder zum Projekt hinzugefügt werden.

Wenn Sie ein Projektelement umbenennen wollen, tun Sie das im Eigenschaftsfenster für das Projektelement. Tabellen werden im Datenmodell-Fenster umbenannt.

3.1.7 Eigenschaften von Projektelementen änderen

Die wichtigsten Eigenschaften von Projektelementen sind Titel und Pfad. Bei allen Projektelementen ausser Tabellen können Sie den Titel frei vergeben. Bei Tabellen muss der Titel mit dem Tabellennamen übereinstimmen, und wir empfehlen dies auch für die anderen Projektelemente. Diese Eigenschaften sind im Eigenschaftsfenster sichtbar und editierbar, wenn das entsprechende Element im Projektfenster selektiert ist.

Eigenschaften für alle Projektelemente:

- *Pfad*: Der Dateiname des Projektelements. Wenn Sie das Projektelement mit dem Explorer umbenennen, müssen Sie hier den Pfad ändern, damit die Verbindung wieder stimmt.
- *Titel*: Sollte mit dem Dateinamen (ohne Verzeichnis und Endung) des Projektelements übereinstimmen, muss aber nicht. Wird in Fenster-Überschriften und Registern angezeigt.

Bei Tabellen gibt es zusätzlich die folgenden Projekt-abhängigen Eigenschaften:

- *Exklusiv: Die Tabelle wird in diesem Projekt so geöffnet, dass keine andere Anwendung darauf zugreifen kann. Diese Option kann nur in der Workgroup-Edition ausgeschaltet werden.*
- *Rechte*: Die Bearbeitungsmöglichkeiten der Tabelle in diesem Projekt. Hier wird definiert, ob Datensätze eingefügt (Neueingabe), Datensätze geändert (Editieren), Datensätze gelöscht (Löschen) oder Indexe angelegt bzw. geändert werden dürfen.
- *Suchpfad*: Wenn die Tabelle nicht in einer Single-File-Datenbank liegt, kann man hier einstellen, wie der Speicherort der Tabelle gefunden wird:
  - *Lokal*: Im privaten Verzeichnis der Anwendung
  - *Projekt*: Unter dem in Pfad angegebenen Dateipfad
  - *Resident*: In dem Verzeichnis von dem aus die Anwendung installiert wurde, z.B. auf der Installations-CD oder -DVD.

Formulare haben diese speziellen Eigenschaften:

- *Rechte*: Hier können Sie abhängig vom Formular festlegen, was mit den Daten geschehen darf. Ein bestimmte Aktion wie beispielsweise das Ändern von Datensätzen, ist nur dann freigeschaltet, wenn das entsprechende Recht sowohl bei der Tabelle als auch im Formular gesetzt ist.
- *Verwendung*: Diese Eigenschaft definiert das Verhalten des Formulars in bestimmten Situationen:
  - *Datensätze bearbeiten*: Das Formular wird im Standard-Menü des User-Modus aufgelistet.
  - *Verknüpfte Datensätze*: Das Formular wird zur Darstellung der verknüpften Datensätze verwendet, wenn der Befehl *Verknüpfte Datensätze anzeigen* oder *Neue verknüpfte Datensätze eingeben* ausgeführt wird.
  - *Schließen zulassen*: Das Formular kann vom Anwender geschlossen werden.
  - *ADL-Panel anzeigen*: Bei der Anzeige als modales Formular wird unten ein Bereich mit Kommentar und einem oder zwei Knöpfen eingeblendet.
  - *FormularbezogeneMakros*: Makros im Formularmodul beziehen sich auf das Formular und nicht auf die Applikation. Mehr dazu unter "Applikations-Module und Formular-Module".

3.1.8 Ein gemeinsames Passwort für alle Tabellen definieren

Wenn ein Projekt mehrere geschützte Tabellen enthält, ist es mühsam, bei jedem Öffnen des Projektes für jede Tabelle einzeln Passwort und Schlüssel einzugeben. Deshalb besteht die Möglichkeit, ein gemeinsames Passwort und einen gemeinsamen Schlüssel für alle Tabellen des Projektes festzulegen.

So definieren Sie ein gemeinsames Passwort:

1. Wählen Sie Projekt/Master-Passwort im Menü des Projektfensters.
2. Geben Sie das gewünschte gemeinsame Passwort und evtl. den gewünschten gemeinsamen Schlüssel ein und bestätigen Sie mit OK.

Wenn die Tabellen des Projektes nur mit Passwort geschützt sind, genügt die Eingabe eines Master-Passwortes. Die zusätzliche Angabe eines Master-Schlüssels bringt weder zusätzliche

Vereinfachung noch zusätzlichen Schutz.

Wenn Tabellen des Projektes mit Schlüssel geschützt sind, genügt die Eingabe eines Master-Passwortes nicht. Da das Master-Passwort zwar die Passwörter der einzelnen Tabellen, jedoch nicht ihre Schlüssel ersetzen kann. In diesem Fall müssen Sie auch noch einen Master-Schlüssel vergeben.

3.2 Datenbestände pflegen

3.2.1 Ein Datenfenster öffnen

Ein Datenfenster ist entweder ein für die betreffende Tabelle erstelltes Formular oder das Tabellenfenster.

Das Tabellenfenster steht immer zur Verfügung und wird durch den Befehl *Ausführen* auf dem Tabellenelement im Projektfenster geöffnet. Auch ein Doppelklick auf die Tabelle im Projektfenster und der Öffnen-Schalter im Projektfenster führen zum selben Ziel.

Ein Datenfenster können Sie im Kontextmenü des Projektfensters mit dem Befehl *Ausführen* öffnen, wenn ein Formular selektiert ist. Auch hier genügt eine Doppelklick auf das Formular oder ein einfacher Klick auf den Schalter für das Öffnen im Projektfenster.

3.2.2 Datensätze betrachten

Sie können Ihre Daten entweder im Tabellenfenster oder in einem Formular betrachten. Das Tabellenfenster steht allerdings nur im Entwicklermodus und nicht im User-Modus zur Verfügung.

Ein Formular kann sowohl eine Felddarstellung als auch eine tabellarische Darstellung enthalten. In der Felddarstellung wird jeweils eine Zeile der Tabelle in Eingabefeldern angezeigt. Die tabellarische Darstellung gewährleistet dagegen einen Überblick über mehrere Datensätze. Oft bietet es sich an beide Darstellungsarten in einem Formular zu kombinieren, eine Formularseite mit einer detaillierten Felddarstellung und eine zweite mit einer Tabellensicht. Das Umschalten zwischen den beiden Darstellungsarten erfolgt mit dem Menübefehl *Ansicht/Formularsicht*. In der Schalterleiste entspricht dem der Knopf mit dem stilisierten Formular. Als Abkürzung dient die Taste F7. Damit *TurboDB Studio* die Formularseite mit der Tabellensicht findet, muss im Formulareditor die Seitennummer angegeben werden.

3.2.3 Datensätze sortiert anzeigen

Die Reihenfolge der Datensätze beim Betrachten richtet sich standardmäßig nach der Abfolge beim Eingeben. Jeder Datensatz hat eine physikalische Satznummer, die seine Zeilennummer in der Tabelle angibt. Wenn Sie Ihre Daten in einer anderen Sortierung darstellen wollen, können Sie eine Sortierordnung auswählen, welche die Reihenfolge bestimmt.

Sortierordnungen können in Indexen abgespeichert werden. Eine eingehende Erläuterung zu diesem Thema finden Sie in den folgenden Abschnitten über die Verwaltung von Indexen.

Nehmen wir an, Sie hätten eine Datei von Kundendaten (KUNDEN) und einen passenden Index über das Feld *Name*, der KNDNAME heisst. Dann können Sie Ihre Kunden in der alphabetischen Reihenfolge anzeigen indem Sie eine Sortierung mit diesem Index vorschreiben.

So stellen Sie eine Sortierordnung ein:

1. Wählen Sie den Menüpunkt *Ansicht/Sortierung* oder klicken Sie den entsprechenden Schalter der Schalterleiste an. Sie erhalten das Dialogfenster zur Auswahl einer Sortierordnung, das eine Liste der vorhandenen Sortierungen enthält.
2. Selektieren Sie mit den Pfeiltasten den gewünschten Index und bestätigen Sie mit *OK*. Die Datensätze werden in der entsprechenden Reihenfolge angezeigt.



Anmerkung:

Falls die gewünschte Sortierordnung nicht vorhanden ist, können Sie mit dem Befehl *Indexe* verwalten im Datenmodell das Dialogfenster für die Index-Verwaltung öffnen um einen neuen Index für diese Sortierung zu erzeugen. Dies ist in dem Abschnitt "Einen Index erstellen" beschrieben.

3.2.4 Neue Datensätze eingeben

Datensätze werden im Datenfenster eingegeben. Um dieses zu öffnen, markieren Sie ein Formular oder eine Tabelle im Projekt und wählen dann den Befehl *Ausführen* im Kontextmenü. Weitere Informationen finden Sie unter "Ein Datenfenster öffnen". Sie können nur dann neue Datensätze eingeben, wenn sowohl bei der Tabelle als auch beim Formular das Recht zur Neueingabe eingetragen ist. Diese Rechte können Sie unter den Eigenschaften der Tabelle bzw. des Formulars im Projektfenster einsehen und modifizieren.

So geben Sie neue Datensätze im Datenfenster ein:

1. Schalten Sie das Datenfenster in den Modus für die Neueingabe, z.B. über den Befehl Bearbeiten/Neue Datensätze eingeben
2. Sie sehen jetzt einen neuen leeren Datensatz und können die einzelnen Felder ausfüllen wie unter "Datensätze ändern" beschrieben.
3. Wenn Sie im letzten Feld des Datensatzes die Eingabetaste drücken oder mit Suchen/Gehe zu/Nächster Datensatz ([Strg]+[Bild ab]) auf den nächsten Datensatz umschalten, wird ein weiterer neuer Datensatz an die Tabelle angehängt. Der Modus für die Neueingabe wird mit dem erneuten Befehl Bearbeiten/Neue Datensätze eingeben beendet. Er wird auch dann beendet, wenn Sie auf einen zurückliegenden Datensatz umschalten, z.B. mit Suchen/Gehe zu/Vorheriger Datensatz.
4. Falls Sie einmal versehentlich einen Datensatz eingefügt haben, den Sie gar nicht haben wollen, wählen Sie Bearbeiten/Rückgängig: Neuer Datensatz oder drücken die Escape-Taste. Nach einer Bestätigung wird der zuviel angefügte Datensatz wieder entfernt, der Neueingabemodus bleibt allerdings aktiv.

3.2.5 Datensätze ändern

Datensätze werden im Datenfenster editiert. Sie können die Datensätze nur dann ändern, wenn sowohl bei der Tabelle als auch beim Formular das Recht zum Editieren eingetragen ist. Diese Rechte können Sie unter den Eigenschaften der Tabelle bzw. des Formulars im Projektfenster einsehen und modifizieren.

So ändern Sie die vorhandenen Datensätze:

1. Schalten Sie mit *Bearbeiten/Datensätze* ändern den Modus für das Ändern der Datensätze ein
2. Sie haben nun die Datensätze im Datenfenster vor sich und können die einzelnen Einträge editieren.
3. Wenn Sie mit dem Editieren fertig sind, beenden Sie den Editiermodus, indem Sie ein zweites Mal den Befehl *Bearbeiten/Datensätze* ändern geben.

Anmerkung:

Falls Menüpunkt und Schalter deaktiviert sind, haben Sie keine Editierberechtigung für die Tabelle. Das kann an mehreren Ursachen liegen.

- In den Formulareigenschaften ist das Recht zum Editieren nicht vergeben, siehe Projektelemente bearbeiten.
- In den Tabelleneigenschaften ist das Recht zum Editieren nicht vergeben, siehe Projektelemente bearbeiten.
- Die Tabellendatei (im Falle eines Datenbank-Verzeichnisses) ist mit der Eigenschaft Nur-Lesen versehen.

Wenn Sie beim Editieren eines Datensatzes bemerken, dass Sie die alten Daten doch noch benötigen, können Sie ihn mit dem Befehl *Bearbeiten/Rückgängig: Datensatz ändern* wiederherstellen.

3.2.6 Datensätze löschen

TurboDB Studio bietet Ihnen für unterschiedliche Situationen mehrere Arten von Löschbefehlen an. Die gelöschten Datensätze werden unter *TurboDB Studio* im Gegensatz zu manchen anderen Datenbanken sofort physikalisch gelöscht. Aus Gründen der Speicherplatz-Ersparnis und der Konsistenz im Netzwerk halten wir dieses Verfahren für vorteilhafter. Es hat allerdings den Nachteil, dass einmal gelöschte Datensätze nicht mehr wiederhergestellt werden können.

So löschen Sie den aktuellen Datensatz:

1. Wählen Sie *Bearbeiten/Datensatz* löschen aus dem Menü oder drücken Sie den Knopf mit dem Mülltonnen-Symbol und bestätigen Sie die anschließende Kontrollabfrage mit *OK*.

So löschen Sie alle Datensätze, für die eine Bedingung zutrifft:

1. Markieren Sie die gewünschten Datensätze in einer Suche mit Bedingung, siehe Datensätze markieren.
2. Wählen Sie den Befehl *Bearbeiten/Markierte Datensätze löschen* aus dem Menü und bestätigen Sie die Kontollabfrage mit *OK*.

So löschen Sie doppelt vorhandene Datensätze:

1. Wählen Sie *Suchen/Doppelte Einträge markieren*, um alle doppelt vorkommenden Datensätze zu markieren
2. Wählen Sie den Befehl *Bearbeiten/Markierte Datensätze löschen* aus dem Menü und bestätigen Sie die Kontrollabfrage mit Ok.

3.2.7 Datensätze markieren

Das Markieren von Datensätzen ist eine nicht in allen Datenbanken übliche Funktion, auf die Sie aber sicher bald nicht mehr verzichten möchten. Sie können Anzeige oder Ausdrucke auf die markierten Datensätze beschränken, die markierten Datensätze sortieren oder alle markierten Datensätze löschen.

So markieren Sie Datensätze manuell:

1. Wählen Sie *Suchen/Markierung umschalten*, um die Markierung des aktuellen Datensatzes ein- und auszuschalten. Der markierte Datensatz wird farbig in der Tabellendarstellung farbig hinterlegt. In der Formulardarstellung erkennen Sie den markierten Datensatz daran, dass der Schalter für die Markierung gekennzeichnet ist.

So markieren Sie Datensätze, die einem Suchkriterium entsprechen:

1. Wählen Sie Suchen/Mit Bedingung aus dem Menü. Es erscheint das Dialogfenster für die Suche mit Bedingung, wo Sie eine Selektion als Such-Bedingung eintragen können. Für kompliziertere Selektionen steht Ihnen der Formel-Assistent zur Verfügung.
2. Unter Aktion markieren Sie entweder Gefundene Datensätze markieren, um alle gefundenen Datensätze zu markieren, Gefundene Datensätze zu den Markierungen hinzufügen, um die gefunden Datensätze zusätzlich zu markieren oder Gefundene Datensätze von den Markierungen wegnehmen, um die gefunden Datensätze von der Markierung auszuschließen. Bestätigen Sie mit *OK* und *TurboDB Studio* führt die gewünschte Aktion für jeden Datensatz aus, der die Such-Bedingung erfüllt.

3.2.8 Nur markierte Datensätze anzeigen

Die Anzeige auf die markierten Datensätze zu beschränken, ist z.B. dann sinnvoll, falls sie in der Kundendatei nur diejenigen sehen wollen, die im Postleitzahlengebiet 8xxxx wohnen, oder wenn Sie sich nur für die Rechnungen interessieren, die noch nicht beglichen sind. In einem solchen Fall, markieren Sie einfach die gewünschten Datensätze über eine Suche mit Bedingung und lassen sich anschließend nur die markierten Datensätze anzeigen.

Um nur die markierten Datensätze anzuzeigen wählen Sie den Menüpunkt *Ansicht/Nur markierte Datensätze*.



3.2.9 Datensätze suchen

Wenn die Datenmengen wachsen, verliert man sehr schnell den Überblick und kann einen gewünschten Datensatz nicht mehr einfach durch Blättern auffinden. Aus diesem Grund gibt es eigene Befehle für die Suche von Datensätzen nach vorgegebenen Kriterien. In TurboDB Studio haben Sie drei leistungsfähige Möglichkeiten, um nach Datensätzen in der Tabelle zu suchen.

Index-Suche

Die erste Methode beruht auf den vorhandenen Sortierordnungen, die in Indexe gespeichert sind. Durch die Sortierung kann ein Datensatz sehr schnell gefunden werden, wenn das Suchkriterium mit den Indexfeldern übereinstimmt.

In einer Kundentabelle mit einem Index für das Datenfeld *Name* kann Frau Schneider in wenigen Augenblicken gefunden werden, auch wenn die Tabelle Zehntausende von Datensätzen enthält.

Wenn für das Suchkriterium ein Index existiert, gehen Sie folgendermaßen vor:

1. Öffnen Sie das Menü Suchen in der Menüzeile des Datenfensters. Die ersten Menüpunkte sind Sortierordnungen zur aktuellen Tabelle. Wenn eine davon dem gewünschten Suchkriterium entspricht, wählen Sie diesen Menüpunkt. Falls die Tabelle viele Indexe besitzt, finden Sie unter dem Menüpunkt Suchen/Andere eine komplette Liste aller Sortierordnungen. Wählen Sie die gewünschte aus.
2. Tragen Sie nun den Suchbegriff ein und bestätigen Sie mit OK. Falls ein passender Datensatz existiert, wird er angezeigt. Ansonsten wird derjenige Datensatz angezeigt, der bezüglich der verwendeten Sortierordnung direkt nach dem gesuchten kommt.
3. Um nach dem nächsten Datensatz mit dem Suchkriterium zu suchen, geben Sie den Befehl Suchen/Weitersuchen, drücken die Taste F3 oder klicken den Schalter mit der Taschenlampe und den drei Punkten.

Wenn keine vordefinierte Sortierordnung für das Suchkriterium besteht können Sie im Datenmodell-Fenster einen neuen Index mit passender Sortierung anlegen. Eine andere Möglichkeit wird im folgenden Absatz beschrieben.

Suche mit Suchbedingung

Die zweite Methode der Suche nach einem Datensatz besteht in der Angabe einer Such-Bedingung. Eine solche Suchbedingung kann sehr einfach sein, wie bei *Name ist "Schneider"*, aber auch sehr kompliziert werden, da mehrere Kriterien verknüpft werden können und anspruchsvolle Funktionen aufgerufen werden dürfen. Die Suche mit einer Bedingung ist gerade bei umfangreichen Datenbeständen langsamer als die Indexsuche, aber dafür sehr flexibel.

So suchen Sie Datensätze, die einer Suchbedingung entsprechen:

1. Wählen Sie *Suchen/Mit Bedingung* aus dem Menü, es erscheint das Dialogfenster für die Suche mit Bedingung zum Eingeben der Such-Bedingung.
2. Geben Sie die gewünschte Selektion wie z.B. *Name ist "Schneider"* ein. Der Formel-Assistent hilft Ihnen bei der Formulierung komplexer Selektionen.
3. Wählen Sie die gewünschte Aktion aus. Wenn Sie nur einen Datensatz suchen, ist die Voreinstellung *Ersten passenden Datensatz selektieren* richtig. Mit den anderen Aktionen können Sie alle passenden Datensätze auf einmal markieren und in mehreren Such-Durchgängen die gewünschten herausfiltern.
4. Falls kein passender Datensatz existiert, gibt TurboDB Studio eine entsprechende Meldung aus. Im anderen Fall wird der erste gefundene Datensatz angezeigt.
5. Zum Suchen des nächsten passenden Eintrags wählen Sie *Suchen/Weitersuchen*, klicken den Schalter mit der Taschenlampe und den drei Punkten oder drücken die Taste *F3*.

Anmerkung:

Selektionen können viele verschiedene Operatoren wie von...bis oder ähnlich enthalten und mit *und*, *oder* und *nicht* verknüpft werden. Um das Eingeben von komplizierten Selektionen zu

vereinfachen, aktiviert der Schalter Assistent im Suchdialog den Formel-Assistenten, wo Sie die Datenfelder und Operatoren nur noch anzuklicken brauchen.

Ein häufiger Fehler beim Eintragen einer Suchbedingung mit Zeichenkette wie z.B. *Name ist "Maier"* besteht darin, dass die Anführungszeichen bei dem gesuchten Namen weggelassen werden.

Volltext-Suche

Wenn Sie für die gewünschten Such-Felder einen Volltext-Index angelegt haben, ist die Suche nach einem Stichwort oft das einfachste und schnellste.

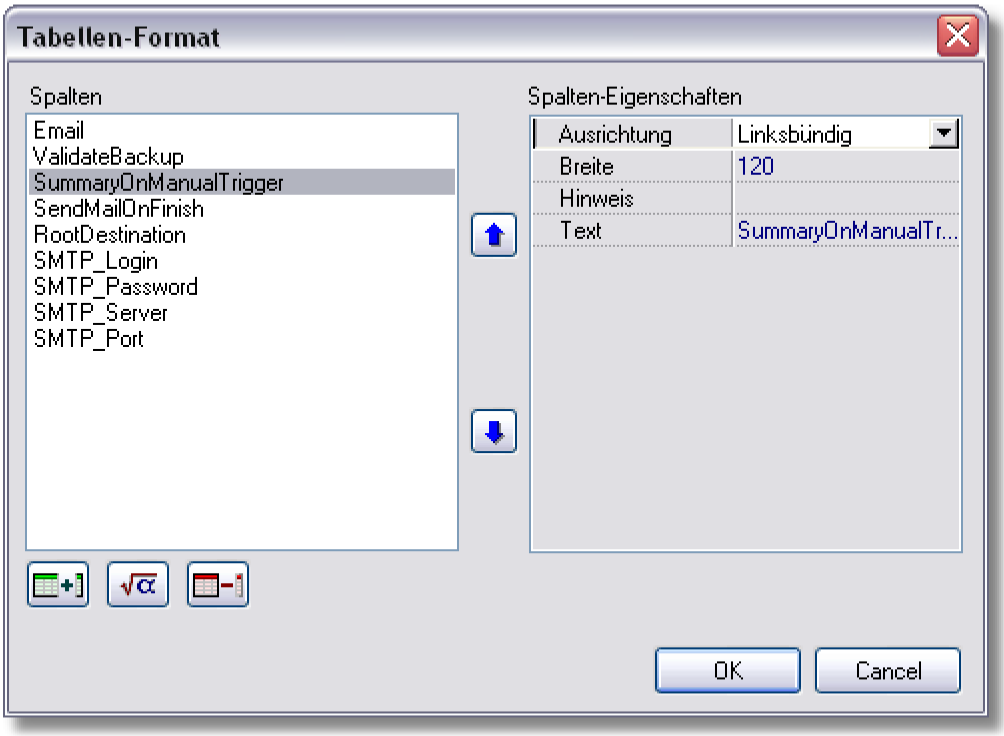
Datensätze nach Stichwörtern suchen, für die ein Volltext-Index erzeugt wurde:

1. Wählen Sie *Suchen/Nach Stichwörtern* aus dem Menü. Sie werden aufgefordert, die zu suchenden Wörter einzutragen.
2. Bestätigen Sie mit OK. Die gefundenen Worte sind im Datenfenster markiert.

3.2.10 Das Tabellenfenster formatieren

Wenn ein Tabellenfenster geöffnet ist, sehen Sie die Datensätze zeilenweise untereinander aufgelistet. Per Vorgabe enthält diese Darstellung eine Spalte für jedes Datenfeld der Tabelle. Sie können die Spaltendarstellung Ihren Wünschen anpassen, indem Sie Spalten löschen, einfügen oder ändern, eine andere Schriftart wählen, oder zum Standard-Format zurückkehren.

Auf diese Optionen greifen Sie über das Dialogfeld Tabellen-Format zu, das Sie mit dem Befehl *Ansicht/Tabellen-Format...* öffnen.



So löschen Sie eine Spalte:

1. Selektieren Sie die gewünschte Spalte.
2. Klicken Sie auf den Schalter zum Löschen (dritter von links).

So fügen Sie eine Spalte ein:

1. Selektieren Sie diejenige Spalte der Tabelle, vor der die neue Spalte eingefügt werden soll.
2. Klicken Sie auf den Schalter zum Einfügen einer neuen Spalte (erster von links). Eine Liste mit möglichen Feldern öffnet sich.
3. Markieren Sie ein Feld oder mehrere aus dieser Liste und drücken Sie Ok.
4. Die markierten Felder werden eingefügt. Sie können nun einzeln ihre Eigenschaften festlegen.

Ausser den aufgeführten Datenfeldern der Tabelle können Sie auch Datenfelder angekoppelter Datensätze anzeigen lassen oder berechnete Spalten definieren.

So fügen Sie eine berechnete Spalte ein:

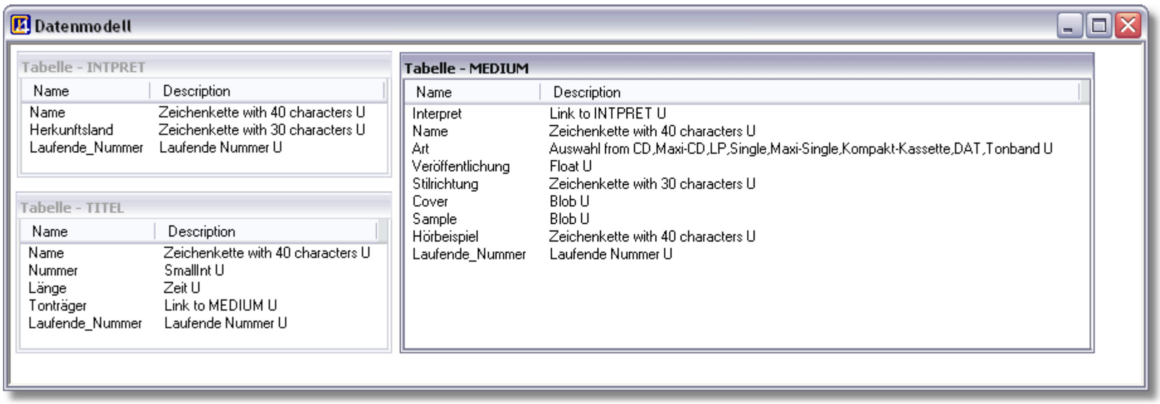
1. Selektieren Sie diejenige Spalte der Tabelle, vor der die neue Spalte eingefügt werden soll.
2. Klicken Sie auf den Schalter zum Einfügen einer neuen berechneten Spalte (zweiter von links). Eine neue Spalte namens <Formel> wird eingetragen.
3. Stellen Sie die Eigenschaften der Formel ein. Dazu geben Sie als Text den entsprechenden Ausdruck in das Eingabefeld ein oder erstellen ihn unter Verwendung des Ausdruck-Assistenten. Für ein Feld einer angekoppelten Tabelle, benutzen Sie die übliche Notation TABELLE.Feld oder Koppelfeld.Feld (Koppelfeld-Notation), z.B: Brutto-Preis - Netto-Preis

3.3 Mit Tabellen und Indexen arbeiten

3.3.1 Mit Tabellen und Indexen arbeiten

Tabellen enthalten die Datensätze und sind damit die grundlegenden Elemente einer Datenbank. Um gesuchte Datensätze auch unter Tausenden schnell zu finden, werden Indexe angelegt, die eine Art sortiertes Inhaltsverzeichnis darstellen.

Tabellen und Indexe bilden die Grundlage der eigentlichen Datenbank. Sie werden im Datenmodell-Fenster bearbeitet, das Sie durch Doppelklick auf das Datenmodell-Symbol im Projektfenster öffnen können.



Im Datenmodellfenster gibt es für jede Tabelle der Datenbank eine Darstellung mit allen Spalten der Tabelle und ihrer Definition.Das U hinter der Definition bedeutet, dass in dieser Spalte leere Einträge zugelassen sind.

3.3.2 Die Tabellen-Formate

TurboDB unterstützt derzeit drei verschiedene Tabellen-Formate genannt Levels, wobei Level 1 der älteste ist und Level 3 der jüngste. Weil im Laufe der Entwicklung neue Funktionalität dazugekommen ist, haben Sie mit Tabellen des Levels 3 die meisten Möglichkeiten, andererseits sind Tabellen mit Level 1 und Level 2 besser kompatibel zu Datenbanken, die mit älteren Werkzeugen der TurboDB-Familie erstellt wurden.

Level 1 Original-Tabellenformat von TDB 5, WinTDB und VDP 1

Level 2 Erweitertes Tabellenformat von VDP 2 und VDP 3

Level 3 Stark erweitertes Tabellenformat von TurboDB, TurboDB.NET und TurboDB Studio

Die folgende Übersicht zeigt die Änderungen in Tabellen mit Level 3 gegenüber der Version Level 2 und Level 1.

Gegenüber Level 2

- Es gibt neue Spaltentypen für 4-Byte und 8-Byte Ganzzahlen
- Es gibt einen neuen Spaltentyp für die Kombination aus Datum und Uhrzeit

- Es gibt neue Spaltentypen für Unicode in String- und in Memo-Feldern
- Es gibt den neuen Spaltentyp für GUIDs.
- Die Tabellen können mit speziellen Sprachtreibern auf die Verwendung von internationalen Sortierordnungen umgestellt werden.

Ihre Vorteile:

- Ganze Zahlen im Bereich -2.000.000.000 bis +2.000.000.000 beziehungsweise noch größer können ohne Genauigkeitsverlust und Performanz-Nachteil abgespeichert werden.
- Der Datum-Zeit-Typ ist ideal für alle Anwendungen wo bisher eine Kombination aus Datumsspalte und Zeitspalte verwendet wurde. Berechnungen zwischen zwei Zeitpunkten werden erheblich vereinfacht.
- Der Datum-Zeit-Typ ist auf Millisekunden genau.
- GUIDs sind in Microsoft-Programmen oft verwendete weltweit eindeutige Bezeichner. Sie stellen eine alternative zu Auto-Nummern dar, wenn man an mehreren Kopien einer Tabelle gleichzeitig ändern möchte.
- Mit den Sprachtreibern ist jetzt z.B. auch eine französische oder polnische Sortierung möglich.

Gegenüber Level 1

- Die Tabellendatei besitzt eine Formatkennung in den ersten Bytes
- Alle Zeichenketten werden in ANSI-Code abgelegt statt in OEM-Code.
- Fließkommazahlen werden im normierten IEEE-Format (8 Byte) abgelegt statt im alten Format (6 Byte).
- Es können beliebig viele Indexe aufgenommen werden, die Länge eines Indexnamens wurde auf 80 erhöht.
- Relationstabellen können ebenfalls lange Dateinamen haben.

Dadurch haben Sie folgende Vorteile:

- Der Zugriff auf Zeichenketten ist schneller, weil eine Konvertierung wegfällt.
- In den Tabellen sind auch Zeichen mit Akzent und anderen Sonderzeichen möglich.
- Lästige Rundungsfehler bei Komma-Zahlen entfallen.
- Die Verarbeitung von Fließkommazahlen wird beschleunigt.
- Der Zahlenbereich für Fließkommazahlen wurde deutlich erweitert.
- Lange Dateinamen werden damit für alle Indexe und Relationstabellen unterstützt.
- Die bisher nötige CNT-Datei bei mehr als fünf Indexen entfällt.

Konvertierung

Aufgrund dieser Vorteile empfehlen wir, bei neu angelegten Tabellen auf jeden Fall das jeweils neueste Format zu verwenden. Sie können aber auch schon vorhandene Tabellen im Tabelleneditor umstellen. Restrukturieren Sie einfach Ihre Tabelle und geben Sie als Tabellenformat den gewünschten Level an. Dabei müssen Sie allerdings folgendes beachten:

- Indexe, die Fließkommazahlen oder Ganzzahlen enthalten, passen nach der Umstellung nicht mehr. Diese Indexe müssen Sie löschen und neu erzeugen. Wiederherstellen genügt nicht.
- Wenn die Tabelle ein Relationsfeld mit mehr als acht Buchstaben im Namen enthält, besteht der Name der zugehörigen Relationstabelle aus den ersten acht Buchstaben des Relationsfeldes. Nach der Umstellung sucht TDB aber nach einer Relationstabelle mit dem vollständigen Namen des Relationsfeldes. D. h. Sie müssen die Relationstabelle umbenennen. Dazu schließen Sie das Projekt und ändern den Namen unter *Werkzeuge/Tabelle* umbenennen...

3.3.3 Eine Tabelle erstellen

Wenn Sie eine neuen Datenbank aufbauen oder eine bestehende erweitern wollen, müssen Sie eine neue Tabelle aufbauen, in der die Datensätze gespeichert werden.

So erstellen Sie eine neue Tabelle für das aktuelle Projekt:

1. Rechtsklicken Sie im Projektfenster und wählen Sie *Neu/Tabelle...* aus dem Kontextmenü.
2. Geben Sie einen Namen für die neue Tabelle ein. Bestätigen Sie die Eingabe mit OK.
3. Nun definieren Sie im Tabellen-Designer den Aufbau der Tabelle, indem Sie Name und Datentyp der einzelnen Spalten festlegen. Bestätigen Sie mit OK, wenn Sie alle Spalten festgelegt haben.
4. Die Tabelle wird erzeugt und ins Projektfenster eingetragen.

Eigenschaften für Tabellenspalten

Name	Name der Spalte. Darf Buchstaben inklusive deutscher Umlaufe, Unterstrich und Bindestrich enthalten. In TurboDB Studio werden bei Spaltennamen Groß- und Kleinschreibung unterschieden.
Datentyp	Einer der verfügbaren Spaltentypen
Formel	Eine Formel, mit der Wert in dieser Spalte automatisch ausgerechnet wird. Nur eintragen, wenn dieser Wert nicht eingegeben werden soll.
Nullable	Wenn dieses Feld angekreuzt ist, darf man Einträge in dieser Spalte leer lassen. Spalten, bei denen dieses Feld gar nicht angezeigt wird, darf man immer leer lassen.
Maximallänge	Bei Zeichenkette die maximale eingebbare Anzahl Zeichen
Verknüpfte Tabelle	Bei Koppel- und Relationsfeldern die mit dieser Spalte verknüpfte Tabelle
Anzeige-Information	Bei Auto-Inkrement-Feldern die Definition für die lesbare Anzeige der Satz-Identität
Eindeutig	Legt bei Auto-Inkrement-Feldern, dass nicht zwei Datensätze mit der selben Anzeige-Information eingetragen werden dürfen.
Werte	Bei Aufzählungstypen die Liste der möglichen Werte. Tragen Sie hier immer einen Wert pro Zeile ein. Für die Werte gelten dieselben Regeln wie für Spaltennamen. Es dürfen keine Sonderzeichen enthalten sein.
Nachkommastellen	Bei Fließkommazahlen die Anzahl der Nachkommastellen. Ist bei Export in Textdatei, dBase-Datei oder XML relevant.

Eigenschaften der Tabelle

Name	Name der Tabelle
Datenbankname	Speicherort der Datenbank
Passwort	Passwort für die Tabelle
Code	Code-Zahl für die Tabelle. Falls angegeben, wird die Tabelle mit dieser Zahl verschlüsselt.
Tabellen-Level	Die Version der Tabellen-Datei, siehe Tabellenformate.
Sprache	Eine dreibuchstabige Kennung für den verwendeten Sprachtreiber.

3.3.4 Eine Tabelle umbenennen

So ändern Sie den Namen einer Tabelle:

1. Öffnen Sie das Datenmodell-Fenster (z.B. durch Doppelklick auf das entsprechende Symbol im Projektfenster)
2. Selektieren Sie die gewünschte Tabelle und wählen Sie Tabelle/Umbenennen... im Menü.
3. Tragen Sie im Dialogfeld den neuen Namen für die Tabelle an und bestätigen Sie mit Ok.

Hinweis

Eine Relationstabelle (d.h. eine Tabelle mit der Endung .rel) kann nicht umbenannt werden, weil

sie immer so heißen muss, wie der Name des Relationsfeldes, zum dem sie gehört.

3.3.5 Spaltentypen

Der Datentyp einer Tabellenspalte bestimmt, welche Art von Informationen in der Spalte gespeichert werden können und wie viel Platz [Byte] ein Eintrag belegt. TurboDB bietet die folgenden Spaltentypen an:

Datentyp	**Speicherbedarf**	**Mögliche Werte**
WideString	2*MaximaleLänge+2	Beliebige Zeichenkette aus Unicode Zeichen
String	Maximale Länge+1	Beliebige Zeichenketten aus Ansi Zeichen
Byte	1	Zahlen zwischen 0 und 255
SmallInt	2	Ganze Zahlen zwischen -32768 und 32767
LongInt	4	Ganze Zahlen zwischen -2147483648 und 2147483647
LargeInt	8	Ganze Zahlen zwischen -9223372036854775808 und 9223372036854775807
Nummerisch/Fließkomma	8	Zahlen zwischen -1.0E38 und 1.0E38
Datum	4	Datumsangaben ab dem 1.1.01
Zeit	2	Zeitangaben von 00:00 bis 23:59
DatumZeit	8	Datum plus Zeit in einem Wert inkl. Sekunden und Millisekunden
Auswahl	1	Die bei der Definition angegebenen Auswahlmöglichkeiten
Ja/Nein	1	JA und NEIN
Kopplung	4	Auto-Nummer des angekoppelten Datensatzes
Relation	0	Mehrere Verknüpfungsinformationen (in *.REL - Datei)
Auto-Nummer	4	Inhalt wird automatisch vergeben (zw. 1 und 2000000000)
Guid	16	128-Bit Identifikation nach Windows Standard
Memo	4	Beliebig lange Memo-Text (in *.MMO -Datei)
WideMemo	4	Beliebig langer Text aus Unicode-Zeichen (in *.BLB - Datei)
Bild	4	Bild- oder Klang-Daten (in *.BLB - Datei)

3.3.6 Die Struktur einer Tabelle ändern

Beim Arbeiten mit einer Tabelle werden Sie immer wieder einmal feststellen, dass ein Feld fehlt oder zu kurz ist. Vielleicht haben Sie auch bei den Wahlmöglichkeiten eines Auswahlfeldes etwas übersehen oder die Dimensionierung eines nummerischen Feldes zu klein gewählt. In allen diesen Fällen müssen Sie den Aufbau der Tabelle ändern. Wenn sie schon viele Datensätze enthält, ist das für das Datenbankprogramm eine umfangreiche Aufgabe, da jeder Satz gelesen und in eine neue Form gebracht werden muss. Für den Benutzer geht es allerdings ganz einfach.

So ändern Sie die Struktur einer Tabelle:

1. Öffnen Sie das Datenmodellfenster. Sie sehen für jede Tabelle der Datenbank den Aufbau.
2. Rechtsklicken Sie auf die gewünschte Tabelle und wählen Sie *Tabellenstruktur*. Im Tabellen-Designer sehen Sie die derzeit vorhandenen Spalten.
3. Fügen Sie neue Spalten hinzu, indem Sie auf den Schalter zum Hinzufügen klicken und die Eigenschaften nach Bedarf einstellen.
4. Editieren Sie vorhandene Spalten, indem Sie die gewünschte Spalte auswählen und die Eigenschaften verändern.
5. Löschen Sie vorhandene Spalten, indem Sie die gewünschte Spalte auswählen und den Schalter zum Löschen anklicken.

6. Ändern Sie die Eigenschaften der Tabelle im Register *Tabelle*.
7. Klicken Sie auf Ok. Sie sehen nun eine Zusammenfassung der Änderungen. Achten Sie hier auf Daten, die gelöscht werden, weil Sie die Spalte entfernt oder gekürzt haben. Diese Daten sind nach dem Restrukturieren verloren.
8. Bestätigen Sie die geplanten Änderungen mit Ok. Die Tabelle wird entsprechend der neuen Struktur angepasst.

Hinweis

Soll einem Auswahlfeld ein weiterer Punkt hinzugefügt werden, ist zu beachten, dass der neue Punkt am Ende der Liste angefügt wird. Ansonsten geht die Zuordnung in den bereits bestehenden Datensätzen verloren.

3.3.7 Tabellen verknüpfen

Tabellen werden verknüpft, um bei Kombinationen der Datensätze nur die zusammengehörenden heraus zu suchen. Eine Rechnungs-Tabelle sollte zum Beispiel so mit der Kunden-Tabelle verknüpft sein, dass zu jeder Rechnung der passende Kunde aus der anderen Tabelle gehört.

TurboDB Studio bietet Ihnen drei verschiedene Möglichkeiten für solche Verknüpfungen. Zum einen durch spezielle Datenfelder (Kopplung und Relation) in den Tabellen, dann durch statische Verknüpfungen in Projekt oder Datenbankjobs und schließlich durch Selektionen beim Ausdruck.

Mit einem Koppelfeld können Sie genau einen Datensatz der angekoppelten Tabelle an einen Datensatz der Haupttabelle ankoppeln. Es handelt sich also um eine n:1-Verknüpfung. Ein Relationsfeld verknüpft jeden Datensatz der Haupttabelle mit beliebig vielen Datensätzen der verknüpften Tabelle. Dazu wird intern eine dritte Tabelle erstellt, die die Verknüpfung mittels zweier Koppelfelder herstellt. Diese Verknüpfung ist eine n:m-Verknüpfung.

Eine statische Verknüpfung besteht in einer erzwungenen Gleichsetzung von Datenfeldern der beteiligten Tabellen. Sie legen z.B. fest: Das Feld *Nummer* der Tabelle KUNDEN soll mit dem Feld *Kunden-Nummer* der Tabelle RECHNUNG übereinstimmen. Diese statischen Verknüpfungen speichern Sie entweder im Projekt oder in der Liste bzw. im Datenbankjob. Statische Verknüpfungen sind die üblichen Verknüpfungen, wie sie auch in anderen Datenbankprogrammen oder in SQL angeboten werden.

Eine Verknüpfung mittels Selektion geschieht durch die Angabe eines Suchkriteriums, das Ausdrücke aus mehreren Tabellen verwendet. Z.B. stellt die Selektion *RECHNUNG.Kundennummer = KUNDEN.Nummer* eine Verknüpfung zwischen den Tabellen RECHNUNG und KUNDEN her.

So verknüpfen Sie Tabellen mit einem Koppel- oder Relationsfeld:

1. Öffnen oder erstellen Sie ein Projekt, in dem sowohl die Haupttabelle als auch die anzukoppelnde Tabelle enthalten sind. Die anzukoppelnde Tabelle muss eine Spalte für Auto-Nummern besitzen. Falls das noch nicht der Fall ist, müssen Sie eine solche Spalte nachträglich hinzufügen.
2. In der Detailtabelle wird auf gleiche Weise ein Koppel- oder Relationsfeld eingefügt. Welches von beiden Sie benutzen hängt davon ab, ob Sie einen oder mehrere Datensätze der angekoppelten Tabelle mit einem Datensatz der Haupttabelle verknüpfen wollen. Im Datenformat des Koppel- oder Relationsfeldes tragen Sie die anzukoppelnde Tabelle ein.
3. Damit sind die beiden Tabellen verknüpft. Wahrscheinlich müssen Sie jetzt noch das Formular der Detailtabelle so ändern, dass das Koppel- oder Relationsfeld enthalten ist. Ein Koppel- oder Relationsfeld verknüpft Tabellen nicht nur für die Auswertung, sondern unterstützt auch die Eingabe von kombinierten Datensätzen.

So erstellen Sie eine ständige statische Verknüpfung zwischen zwei Tabellen des gleichen Projektes:

1. Wählen Sie *Tabelle/Tabellen verknüpfen...* aus dem Menü des Datenmodellfensters, um das Dialogfeld für Tabellen-Verknüpfungen anzuzeigen.
2. Geben Sie die gewünschten Gleichsetzungen zwischen den Tabellen ein und bestätigen Sie mit OK.
3. Damit sind die Tabellen verknüpft. Den Effekt der statischen Verknüpfung können Sie am besten beobachten, wenn Sie einen Datenbankjob mit einer Liste der Felder aus beiden Tabellen ausdrucken lassen.

So erstellen Sie statische Verknüpfungen während der Ausführung eines Datenbankjobs:

1. Öffnen Sie den Texteditor für den gewünschten Datenbankjob.
2. Fügen Sie, falls noch nicht vorhanden, vor allen anderen Bereichen einen Bereich .prolog ein.
3. Tragen Sie in diesem Bereich nach dem Kommando .relation die gewünschten statischen Verknüpfungen durch Komma getrennt ein.

Hinweis

Die so festgelegten statischen Verknüpfungen gelten nur während des Ausdrucks des Datenbankjobs und werden anschließend wieder entfernt.

3.3.8 Das ADL-System

Primärtabelle

Ein Formular, ein Bericht oder ein Datenbankjob arbeiten also immer auf einer Menge von (teilweise) verknüpften Datenbank-Tabelle. Eine dieser Tabellen wird als Haupttabelle betrachtet und heißt dann Primärtabelle:

- Im Formular ist die Primärtabelle diejenige, in der man sich mit den Navigationstasten der Werkzeugleiste bewegen kann.
- Im Bericht ist die Primärtabelle die Tabelle Hauptdaten-Bereichs.
- Im Datenbankjob ist die Primärtabelle diejenige, deren Datensätze im Datenbereich abgearbeitet werden.
- Im Modul ist spielt die Primärtabelle kein große Rolle.

Welche Tabelle Primärtabelle ist, bestimmt der Projektordner, in dem sich das Formular, der Datenbankjob oder der Bericht befindet. In Datenbankjobs kann die Primärtabelle mit dem Kommando *primärdatei* oder *primtableis* umgestellt werden.

3.3.9 Einen Index erstellen

Ein Index ist ein sortiertes Verzeichnis aller Datensätze einer Tabelle, das in einer eigenen Datei abgelegt ist. Mit Hilfe des Index können Sie die Reihenfolge der Darstellung Ihrer Datensätze festlegen und die Suche nach bestimmten Datensätzen erheblich beschleunigen.

So erstellen Sie einen neuen Index

1. Öffnen Sie das Datenmodell-Fenster durch Doppelklick auf das entsprechende Symbol im Projektfenster. Sie sehen eine Darstellung aller Tabellen der Datenbank mit den Feldern.
2. Rechtsklicken Sie auf die Tabelle, für die Sie einen Index erstellen wollen und wählen Sie *Indexe...* Sie sehen nun die Indexverwaltung.
3. Klicken Sie auf den Schalter zum Hinzufügen eines neuen Index, um das Index-Definitionsfenster zu öffnen.
4. Hier vergeben Sie einen Namen für den neuen Index und wählen zwischen einem hierarchischen Index, der aus einer Auflistung von Tabellenfeldern besteht und einem berechneten Index, der durch eine Formel definiert ist. Definieren Sie, ob in dem Index mehrere Einträge mit derselben Index-Information vorhanden sein dürfen.
5. Spezifizieren Sie einen hierarchischen Index, indem Sie die gewünschten Spalten auflisten. Bei jeder Spalte können Sie angeben, ob sie aufsteigend oder absteigend sortiert werden soll.
6. Spezifizieren Sie einen berechneten Index, indem Sie eine Formel zur Berechnung der Index-Information angeben.
7. Bestätigen Sie im Index-Defnitionsfenster mit Ok. Der Index wird erzeugt und zur Tabelle hinzugefügt.

3.3.10 Einen Index löschen

Unter Umständen möchten Sie einen nicht mehr benötigten Index löschen, um Speicherplatz zu sparen und um das Einfügen von Datensätzen zu beschleunigen. Durch das Löschen eines Index gehen keine Informationen verloren. Sie können den Index jederzeit wieder aufbauen, allerdings dauert das unter Umständen einige Zeit.

So löschen Sie einen Index

1. Öffnen Sie das Datenmodellfenster mit einem Doppelklick auf das entsprechende Symbol im Projektfenster.
2. Rechtsklicken Sie auf die gewünschte Tabelle und wählen Sie *Indexe...* aus dem Menü, um die Index-Verwaltung zu öffnen.
3. Selektieren Sie den Index, den Sie löschen möchten und klicken Sie auf den Schalter zum Löschen des Index.

Hinweis

Bestimmte vom System benötigte Indexe können Sie nicht löschen. Dazu gehören der ID-Index und der Auto-Nummer-Index (INR).

3.3.11 Einen Index reparieren

Gerade während der Entwicklung kann es passieren, dass ein Index beschädigt wird, zum Beispiel wenn Ihr Programm abstürzt. Sie erkennen das daran, dass beim Zugriff über diesen Index Datensätze zu fehlen scheinen, die bei anderen Sortierungen aber vorhanden sind.

So reparieren Sie einen beschädigten Index:

1. Öffnen Sie das Datenmodellfenster durch einen Doppelklick auf das entsprechende Symbol im Projektfenster.
2. Selektieren Sie die Tabelle, zu welcher der beschädigte Index gehört.
3. Wählen Sie *Tabelle/Indexe...* aus dem Menü, um die Index-Verwaltung zu öffnen.
4. Selektieren Sie den beschädigten Index in der Liste der Indexe.
5. Klicken Sie auf den Schalter zum Reparieren des Index.

So reparieren Sie alle Indexe einer Tabelle:

1. Öffnen Sie das Datenmodellfenster durch einen Doppelklick auf das entsprechende Symbol im Projektfenster.
2. Selektieren Sie die Tabelle, deren Indexe Sie reparieren möchten.
3. Wählen Sie *Tabelle/Indexe...* aus dem Menü, um die Index-Verwaltung zu öffnen.
4. Klicken Sie auf den Schalter zum Reparieren aller Indexe.

3.3.12 Volltext-Indexe

Bisher zeichneten sich Indizierung bei Datenbanksystemen dadurch aus, dass eine Tabelle über ein oder mehrere Felder in eine entsprechende Reihenfolge gebracht wurde. Übliche Sortierungen waren hierbei z.B. bei Adressdaten der Name, Vorname, Ort bzw. Kombinationen daraus. Entscheidend dabei ist, dass der Index für jeden Datensatz in einer strengen Vorschrift ermittelt werden muss. Diese Indexe sind weiterhin vorhanden und helfen in den gängigen Suchsituationen auch sehr gut weiter.

Darüber hinaus gibt es bei Tabellen auch Felder die mangels geeigneter strikter Regeln nicht für den Aufbau eines Index geeignet waren. Bekanntestes Beispiel dafür sind Memo-Felder, die es erlauben Text in mehr oder weniger unbegrenzter Menge einzutragen. Doch auch bei "normalen" alphanummerischen Feldern wurden in manchen Fällen mehrere Einträge gemacht. Bekanntestes Beispiel ist hier eine Literaturverwaltung, bei der im Feld Autoren sämtliche am Buch beteiligten Autoren mit Komma getrennt aufgeführt wurden. Sie können sich vorstellen, dass es in diesen Fällen nicht oder nur sehr begrenzt möglich war einen geeigneten Index zu erstellen.

Durch die Einführung des Volltextindex sind diese Situationen nun sehr einfach zu beherrschen. Im Menü *Tabelle/Volltext-Index erstellen...* ändern Sie durch einen Assistenten geführt alle für diesen Zweck notwendigen Angaben. Dazu gehören die Angabe einer zusätzlichen Tabelle, die

die eigentlichen Suchdaten enthält, sowie die Auswahl der Felder, die in den Index aufgenommen werden sollen. Optional können Sie in einer externen Datei alle Wörter definieren, die nicht in den Index aufgenommen werden sollen. Nach dem Fertigstellen des Index können Sie in Ihrer Tabelle nun auch im Volltext suchen. Entsprechend der üblichen Suche mit Bedingung können ein oder mehrere Suchbegriffe abgefragt werden, eine Angabe des Datenfeldes ist jedoch nicht mehr nötig.

Hinweis

Volltextindexe werden **nicht** automatisch gewartet! Bei Datenänderungen in der Tabelle muss der Index neu aufgebaut werden, wobei die meisten Einstellungen vom Pfadfinder übernommen werden.

Siehe auch

Volltextsuche in der TurboPL Referenz

3.4 Performanz und Mengengerüst

3.4.1 Datenbankgröße erhöhen

TurboDB kann prinzipiell bis zu maximal 2 Mrd. Datensätze pro Tabelle verwalten. Je nach den Einstellungen bei den einzelnen Datenbank-Objekten kann es aber sein, dass schon vorher eine Fehlermeldung ausgelöst wird, dass die Kapazität der Tabelle nicht ausreicht, um die gewünschte Anzahl an Datensätzen einzutragen. Falls das passiert, sollten Sie auf folgende Einstellungen achten:

Die Indexseiten-Größe ist die häufigste Beschränkung für die Anzahl der Datensätze in der Tabelle. TurboDB verwaltet bis zu 32767 Index-Seiten pro Index. Dadurch ergibt sich eine maximale Kapazität des Index von *Ordnung* * 2 * 32767. Wenn Sie zum Beispiel einen Index definieren, von dem ein Eintrag 256 Bytes benötigt und dafür eine Seitengröße von 4 kB vorsehen, dann passen 16 Einträge auf eine Seite, wodurch sich eine maximale Anzahl von 16 * 32767 = 524.272 ergibt. Um die maximale Anzahl an Datensätze zu erreichen, müssen Sie die Indexseite erheblich größer wählen, nämlich 16 MB.

Wenn die Tabelle tatsächlich 2 Mrd. Datensätze aufnehmen soll, kommen normale Record-Ids für die Datensätze nicht mehr in Frage, das diese ebenfalls nur einen Zahlenbereich bis 2 Mrd haben. D.h. wenn der ersten Datensatz gelöscht und ein neuer angefügt wird, wird schon der Zahlenbereich für die Record-Id überschritten. Deshalb benötigen Sie für solche Tabellen lange Auto-Nummern, die einen praktisch unbegrenzten Zahlenbereich bis 9.223.372.036.854.775.807 haben.

Außer der Tabelle selbst, kann auch die Menge der Memo und/oder Blob-Daten eine Beschränkung darstellen. Und auch hier hängt die Kapazität von der Seitengröße dieser Dateien ab, weil die maximale Anzahl an Seiten 268 Mio. beträgt. Bei einer maximalen Seitengröße von 64 KB ergibt sich hieraus eine Obergrenze von 17,5 TB für alle Blobs und Memos einer Tabelle. Memos im Ansi-Zeichensatz zählen dabei extra, d.h. Sie können maximal 17,5 TB Blobs plus 17,5 TB Ansi-Memos abspeichern. Dies gilt allerdings wiederum nur bei der maximalen Seitengröße. Wenn Sie eine geringere Seitengröße als 64 KB wählen, reduziert sich die maximale Kapazität entsprechend.

Das Tabellenformat wiederum spielt für die Anzahl der Datensätze keine Rolle. Derzeit können alle Versionen bis zu 2 Mrd. Datensätze aufnehmen.

Anders als in früheren Versionen von TurboDB ist die Tabellengröße in Byte keiner realen Beschränkung mehr unterlegen (sie beträgt 9 Mio. TB). In den Versionen vor 4.0 war die maximale Größe einer Tabelle auf 2 GB beschränkt, was die reale maximale Anzahl an Datensätzen erheblich reduziert hatte.

TurboDB kann außerdem bis zu 254 Tabellen in einer Datenbank verwalten. Falls diese Grenze überschritten wird, müssen die Tabellen auf zwei Datenbanken aufgeteilt werden.

3.4.2 Speicherverbrauch reduzieren

In manchen Anwendungen ist es wichtig, dass der Hauptspeicherverbrauch eine bestimmte Grenze nicht überschreitet. Dazu können folgende Hinweise nützlich sein.

Die wichtigste bestimmende Größe für den Hauptspeicherverbrauch ist die Cache-Größe der Datenbank. Wenn Sie hier einen niedrigeren Wert einstellen, erhalten Sie sofort auch einen geringeren Hauptspeicherverbrauch. Allerdings kann sie eine zu kleine Cache-Größe negativ auf den Durchsatz der Anwendung auswirken, weil dann nicht mehr soviel gecacht werden kann.

3.4.3 Datenbank verkleinern

Wenn Sie die Größe der Datenbank auf der Festplatte reduzieren wollen, beachten Sie bitte folgende Regeln:

Speichern Sie keine redundanten Daten in der Datenbank. Dies kommt nicht nur dem Platzbedarf zugute sondern auch der Performanz und der Daten-Integrität.

Achten Sie darauf, dass die Datenbank-Spalten, insbesondere die Strings, nur die wirklich benötigte Größe haben. Wenn Sie z.B. mit 128 Zeichen statt 255 auskommen, dann haben Sie bei 20 Mio Datensätzen 2,5 GB gespart.

Die Indexe und die Datenbank-Tabellen ab Version 4.0 haben einen Füllgrad von 50%. D.h. bis zur Hälfte der Dateigröße ist nicht benutzter Platz. Sie können diese Verschwendung reduzieren, indem Sie die Tabelle restrukturieren und die Indexe auffrischen. Dadurch wird der Füllgrad der Datenbank-Objekte erhöht. Dieser höhere Füllgrad kann dann bei intensiver Nutzung wieder auf 50% sinken. Deshalb ist dieser Rat in erster Linie für Datenbank sinnvoll, die nach der Auslieferung nicht mehr geändert werden (Katalog, Verzeichnisse usw.).

3.4.4 Den Durchsatz verbessern

Wenn sie die Geschwindigkeit von Datenbank-Operationen erhöhen möchten, sollen Sie folgende Möglichkeiten in Betracht ziehen:

Um die Geschwindigkeit von Abfragen zu erhöhen sind zusätzliche Indexe meist die erste Wahl. Wenn in der Anwendung häufig bestimmte Werte von Spalte1 gesucht werden, dann wird ein Index über Spalte1 diese Suche erheblich beschleunigen vorausgesetzt, die Tabelle enthält eine größere Anzahl an Datensätzen.

Manchmal ist auch das Gegenteil davon, nämlich Indexe zu entfernen, die richtige Optimierung. Indexe beschleunigen nämlich die Suche, verlangsamen aber alle Änderungs-Operationen auf der Tabelle. Falls das Einfügen, das Ändern oder das Löschen in der Tabelle die kritischen Operationen in Ihrer Anwendung sind und Ihre Tabellen viele Index haben, könnten Sie versuchen einen Großteil der Indexe zu löschen.

Je nachdem, welche Operationen in Ihrer Anwendung überwiegen, sind die verschiedenen Tabellenformate von TurboDB unterschiedlich optimiert. Die alten Tabellenformate bis 3.0 sind sehr schnell beim Anfügen und auch beim Ändern von Datensätzen aber eher langsam beim Löschen. Die neuen Tabellenformate ab 4.0 haben ein eher ausgeglichenes Zeitverhalten und sind damit beim Einfügen und Ändern eher etwas langsamer, beim Löschen aber dafür deutlich schneller. Beim Auffinden von Datensätzen sind die alten Datenformate besonders schnell, wenn eine Satznummer angegeben wird. Die neuen Formate verwenden die Record-Id als Satznummer und können daher zusätzlich auch angekoppelte Datensätze schneller finden.

Beim Arbeiten mit SQL-Abfragen gibt es eine ganze Reihe von Tipps, wie die Performanz gesteigert werden kann:

- Wenn die *where*-Klausel mehrere Bedingungen enthält, beginnen Sie mit derjenigen, die auf die wenigsten Datensätze zutrifft. Alternativ schreiben Sie diejenige Bedingung zuerst, für die es einen Index gibt.
- Vermeiden Sie überflüssige Klammern.
- Ziehen Sie den *join* einer Verknüpfung in der *where*-Klausel vor. TurboDB versucht zwar *where* -Bedingungen soweit wie möglich in Joins zu konvertieren, bei komplizierten Abfragen kann das aber evtl. nicht gelingen.
- Verwenden Sie keine Unterabfragen, wenn auch eine komplexere einstufige Abfrage das

Ergebnis liefern kann.

Wenn Sie umfangreiche Operationen auf einer Datenbank-Tabelle durchführen, sollten Sie überlegen, eine explizite Sperre auf diese Tabelle zu setzen. Durch diese Sperre, können andere Applikationen in dieser Zeit zwar nicht auf die Tabelle zugreifen (deshalb sollte die Operation nicht länger als ein paar Sekunden dauern), die Operation selbst wird aber erheblich rascher durchgeführt, weil überflüssige Schreib- und Sperraktionen unterbleiben.

Verwenden Sie TurboPL Prozeduren für komplexe Datenbank-Operationen. TurboPL Prozeduren sind von sich aus schon schneller als der Code in der Programmiersprache der Anwendung. Zusätzlich enthält TurboPL noch Befehle zum Optimieren der Operation, die es in der Programmiersprache der Anwendung gar nicht gibt. Dazu gehören insbesondere die schnellen Tabelle-Scans mit *LoopRecs* und *sub* sowie das Arbeiten mit Markierungen.

Bei der Arbeit mit TurboPL sollten die Programme natürlich wo nötig auf Geschwindigkeit optimiert sein, auch wenn es nicht um Datenbank-Operationen geht. Hinweise dazu finden Sie im nächsten Abschnitt.

3.4.5 TurboPL Programme optimieren

In diesem Abschnitt finden Sie Tipps, wie Sie die Ausführung von TurboPL-Programmen schneller machen können.

LoopRecs und SUB verwenden statt ReadRec/NextRec

Wenn Sie eine Schleife über alle Datensätze einer Tabelle ausführen möchten, können Sie das mit der Funktion *LoopRecs* oder dem Kommando *SUB* schneller erzielen.

Unnötige Berechnung von Statistik-Funktionen vermeiden

Schleifen über Datensätze berechnen normalerweise alle Statistik-Funktionen im ganzen Programm neu. Da kommen bei großen Projekten leicht einige hundert Statistik-Funktionen zusammen, wodurch eine Schleife über beispielsweise 100.000 Datensätze deutlich gebremst wird. Mit dem *aggregates*-Kommando können Sie die Berechnung der Statistik-Funktionen einschränken.

Globale Variablen verwenden statt Datenbankfelder

Ab TurboDB Studio sind globale Variablen möglich, die vom Start des Programms bis zum Ende erhalten bleiben. Wenn bestimmte Werte zwar allgemein verfügbar aber nicht gespeichert sein sollen, können Sie deshalb solche globalen Variablen einsetzen statt Werte in der Systemtabelle zu speichern. Der Zugriff ist hier wesentlich schneller.

String-Operationen aus Schleifen herausnehmen

Seit TurboDB Studio entfällt die Längenbeschränkung von 255 Zeichen für Zeichenketten. Allerdings können Operationen wie *Scan* auf sehr langen Zeichenketten (etliche Megabyte) lange dauern und sollten deshalb nicht in Schleifen eingesetzt werden.

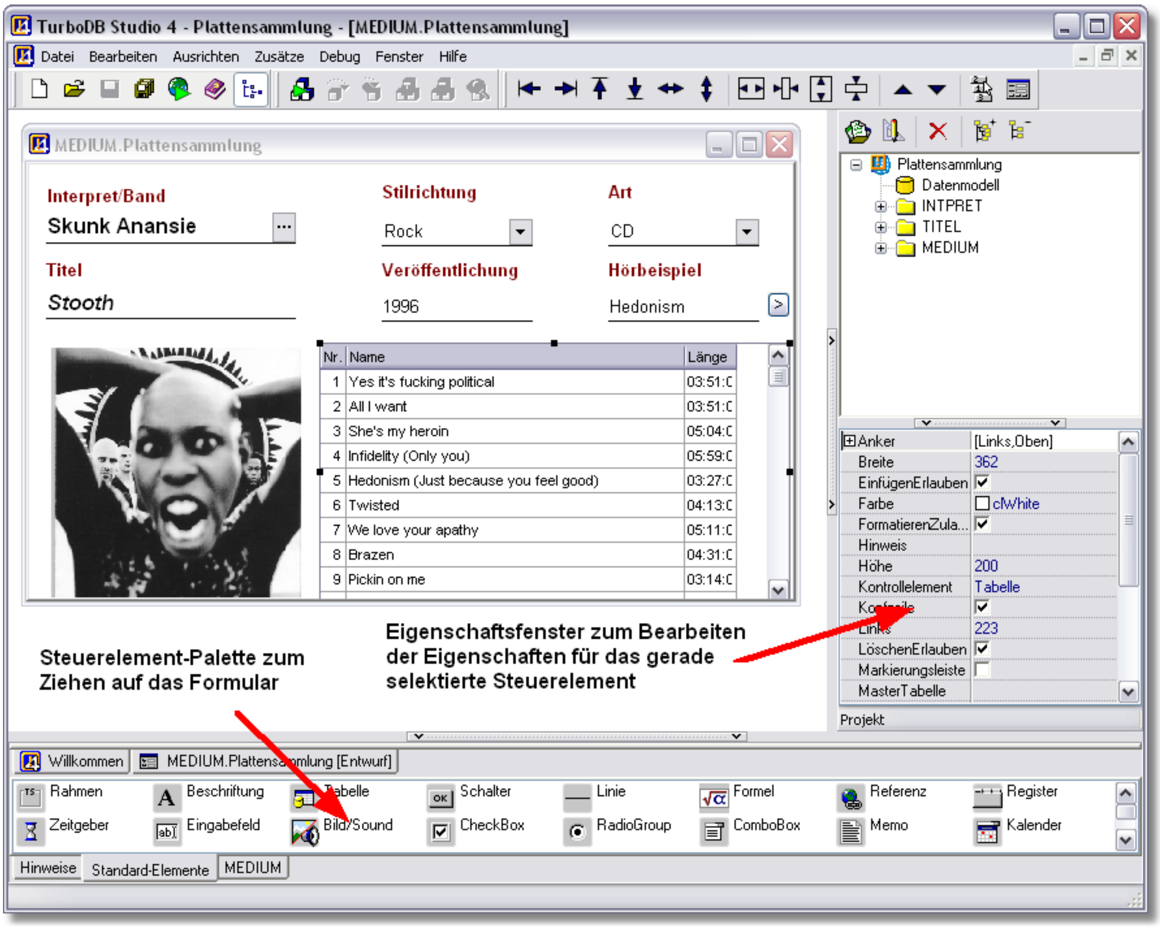
Sehr lange Strings im RamText bearbeiten

Insbesondere das wiederholte Anhängen von Text an eine sehr lange Zeichenkette (mehrere Megabyte) kann langsam sein. Wenn Sie die Zeichenkette in den *RamText* schreiben und dort die Anhänge-Operationen durchführen und den String dann wieder aus dem *RamText* lesen, ist das unter Umständen erheblich schneller.

3.5 Formulare gestalten

3.5.1 Formulare gestalten

Formulare werden im Formulareditor entworfen und gestaltet. Sie sehen hier das Formular so, wie etwas später aussehen wird und können Steuerelemente auf das Formular ziehen, Position und Größe einstellen sowie die Eigenschaften setzen.



So gestalten Sie ein Formular

1. Öffnen Sie den Formular-Designer, indem Sie das Formular im Projektfenster selektieren und dann im Kontextmenü *Entwerfen* auswählen.
2. Ziehen Sie Steuerelemente aus den Paletten auf das Formular. Sie müssen dabei das Steuerelement anklicken, die linke Maustaste gedrückt halten und auf die gewünschte Stelle im Formular ziehen.
3. Stellen Sie die Eigenschaften für das Steuerelement im Eigenschaftsfenster ein.
4. Testen Sie das Formular über den Menüpunkt *Zusätze/Formular testen...*

Eigenschaften des Formulars

BeimÖffnen	Ein Makro, das beim Öffnen des Formulars ausgeführt wird, nachdem der erste Datensatz angezeigt wurde.
BeimSchließen	Ein Makro, das beim Schließen des Formulars ausgeführt wird, bevor die Datenbanktabelle geschlossen wird.
BeimVerlassen	Ein Makro, das beim Datensatzwechsel im Editiermodus ausgeführt wird, bevor der aktuellen Datensatz in die Datenbank geschrieben wird.
NachDemVerlassen	Ein Makro, das beim Datensatzwechesel im Editiermodus ausgeführt wird, nachdem der aktuelle Datensatz in die Datenbank geschrieben wurde.

BeimBetreten	Ein Makro, das beim Datensatzwechsel im Editiermodus ausgeführt wird, nachdem der neue Datensatz gerade angezeigt wurde.
Breite	Breite des Formulars in Pixeln
Höhe	Höhe des Formulars in Pixeln
Farbe	Farbe des Formular-Hintergrunds
Gültigkeitsüberprüfung	Eine Bedingung, die erfüllt sein muss, damit ein neuer oder geänderter Datensatz in die Datenbank geschrieben werden kann.
UngültigkeitsMeldung	Ein Text, der angezeigt wird, wenn die Gültigkeitsüberprüfung fehlschlägt.
TabellarischeSicht	Die Nummer derjenigen Seite im Hauptregister, die angezeigt wird, wenn im Menü *Ansicht/Formularsicht* ausgeschaltet wird. Damit kann man dem Anwender eine einfach Umschaltung zwischen tabellarischer Ansicht und Formularansicht anbieten.

Siehe auch
Mehrseitige Formular erstellen

3.5.2 Selektieren im Formulareditor

Wenn Sie ein Element im Formulareditor selektieren wollen, klicken Sie es einfach mit der linken Maustaste an. Es wird dann mit Markierungen zum Verändern der Größe versehen und seine Eigenschaften werden im Eigenschaftsfenster aufgelistet.

Wenn Sie mehrere Elemente auf einmal selektieren wollen, gibt es zwei Möglichkeiten:

- Sie klicken die Elemente mit der linken Maustaste an, während Sie die Strg-Taste gedrückt halten.
- Sie ziehen einen Rahmen mit der Maus auf, während Sie die Strg-Taste gedrückt halten.

3.5.3 Steuerelemente ausrichten

Um ein ansprechendes Formular zu erhalten, werden Sie die einzelnen Elemente oft bündig aneinander ausrichten wollen. Dazu gibt es zwei Möglichkeiten. Entweder Sie verwenden das Raster oder Sie benutzen die verschiedenen Befehle zum Ausrichten von Steuerelementen.

So aktivieren Sie das Raster zum Ausrichten von Steuerlementen

1. Wählen Sie *Zusätze/Rastereinstellungen...* im Menü.
2. Markieren Sie das Feld *An Rasterpunkten einrasten*.
3. Falls gewünscht, markieren Sie das Feld *Rasterpunkte anzeigen*. Diese Option hat allerdings nur dann eine sichtbare Wirkung, wenn nicht ein Register das Formular vollständig bedeckt.
4. Bestätigen Sie das Dialogfeld mit *OK*.

Beim Verschieben und Vergrößern bewegen sich die Steuerelement jetzt in der angegebenen Schrittweite. Dadurch ist eine exakte Ausrichtung auf dem Formular sehr einfach.

So richten Sie die selektierten Steuerelemente bündig aus

1. Selektieren Sie die auszurichtenden Steuerelemente durch Anklicken bei gedrückter Umschalttaste.
2. Wählen Sie im Menü *Ausrichten/Position/<Richtung>*.
3. Alle selektierten Steuerelemente werden nun soweit in die ausgewählte Richtung verschoben, bis alle genau in einer Linie liegen.

So passen Sie die Größe der selektierten Steuerelement an

1. Selektieren Sie die auszurichtenden Steuerelemente durch Anklicken bei gedrückter Umschalttaste.
2. Wählen Sie im Menü *Ausrichten/Größe/<Änderung>*.
3. Alle selektierten Steuerelemente erhalten nun einen neue Größe entsprechend der ausgewählten Änderung.

3.5.4 Steuerelemente einstellen

Beim Einstellen der Steuerelemente legen Sie seine Eigenschaften im Eigenschaftsfenster fest. Es gibt Eigenschaften, die jedes Steuerelement aufweist:

Das sind die Eigenschaften, die alle Steuerelemente besitzen:

Anker	Legt fest, welche Ecken des Steuerelements sich beim Vergrößern des Formulars mitbewegen.
Breite	Die Breite des Steuerelements in Pixeln
Farbe	Die (Hintergrund-) Farbe des Steuerelements
Hinweis	Ein Text, der in einem gelben Fähnchen angezeigt wird, wenn der Anwender die Maus über dem Steuerelement ruhen lässt
Höhe	Die Höhe des Steuerelements in Pixeln
Klasse	Der Typ des Steuerelements
Links	Die linke Position des Steuerelements in Pixeln relativ zu seinem Elternelement
Name	Frei wählbarer Name für das Steuerelement. Wenn Sie es in TurboPL ansprechen wollen, müssen Sie jedoch einen gültigen TurboPL-Bezeichner (ohne Sonderzeichen, keine Ziffer am Anfang) eintragen.
Oben	Die obere Position des Steuerelements in Pixeln relativ zu seinem Elternelement
Rahmen	Der Rahmen für das Steuerelement. Steht der Rahmen auf Standard, wird unter Windows XP der eingestellte Stil verwendet. Steht der Rahmen auf *Ultraflach* und die Farbe des Elementes auf *clDefault*, dann ändert ein Schalter seine Farbe, wenn man mit der Maus darüber fährt.
Sichtbar	Legt fest, ob das Steuerelement bei der Ausführung angezeigt wird.
Transparent	Legt fest, ob der Hintergrund des Steuerelements durchsichtig ist.

Diese Eigenschaften haben alle Steuerelemente, die Text anzeigen:

AnzahlZeilen	Legt fest, wieviele Zeilen Text im Steuerelement angezeigt werden sollen. Dadurch wird die Höhe des Steuerelements automatisch bestimmt und kann dann nicht geändert werden. Wenn Sie die die Schriftart ändert und die Anzahl der Zeilen gleich bleibt, kann das Steuerelement seine Höhe ändern, wenn die neue Schrift kleiner ist als die alte.
Ausrichtung	Wie bündig der Text innerhalb des Steuerelements angeordnet wird
Schriftart	Die Schriftart für das Steuerelement. Siehe auch "Die Schriftart eines Steuerelements ändern".

Eigenschaften für alle Eingabefelder

Aktivierungsbedingung	Eine Bedingung, die erfüllt sein muss, damit das Steuerelement aktiviert ist, d.h. damit der Anwender darin editieren kann.
BeimBetreten	Ein Makro, das ausgeführt wird, wenn das Steuerelement den Fokus erhält.
BeimVerlassen	Ein Makro, das ausgeführt wird, wenn das Steuerelement den Fokus verliert.
BeimÄndern	Ein Makro, das ausgeführt wird, wenn der Anwender den Inhalt des Eingabeelements ändert.
Feldname	Der Name der Tabellenspalte, deren Daten in diesem Steuerelement dargestellt werden
Gültigkeitsbedingung	Eine Bedingung, die erfüllt sein muss, damit die Eingabe des Anwenders in das Eingabeelement akzeptiert wird.
MaximalerWert	Der höchste Wert, der in das Eingabefeld eingetragen werden darf.
MeldungBeiGültigkeitsverletz	Der Text, der angezeigt wird, wenn die Gültigkeitsbedingung nicht

ung	erfüllt ist.
MinimalerWert	Der niedrigste Wert, der in das Eingabefeld eingetragen werden darf.
Muster	Ein Schema, nach dem die Werte in das Eingabefeld eingetragen werden müssen.
NurNeueingabe	Der Anwender kann das Eingabefeld nur bei einem neuen Datensatz editieren.
VerdeckteEingabe	Legt fest, ob der Text im Eingabefeld als Sternchen angezeigt werden soll (für Passworteingabe).
Vorgabe	Ein Ausdruck, mit dem das Eingabefeld beim Anlegen eines neuen Datensatzes vorbelegt wird.

Spezielle Eigenschaften für das Tabellenelement

EinfügenErlauben	Der Anwender kann im Tabellenelement neue Datensätze eintragen.
FormatierenZulassen	Der Anwender kann die Spalten des Tabellenelements vergrößern und verschieben.
Kopfzeile	Die Überschriften für die Spalten werden angezeigt.
LöschenErlauben	Der Anwender kann Datensätze im Tabellenelement löschen.
MarkierungLeiste	Auf der linken Seite des Tabellenelements ist eine Leiste zu sehen, wo markierte Datensätze sichtbar sind.
MasterTabelle	Der Name der übergeordneten Datenbank-Tabelle, für die verknüpfte Datensätze angezeigt werden
Rollbalken	Legt fest, ob auf der rechten Seite des Tabellenelements ein Rollbalken angezeigt wird.
Sortierung	Eine Index-Definition, die festlegt, wie die Einträge im Tabellenelement sortiert werden.
Tabelle	Die Datenbank-Tabelle, deren Daten im Tabellenelement dargestellt werden.
ÄndernErlauben	Der Anwender kann Datensätze im Tabellenelement verändern.

Spezielle Eigenschaften für das Formelelement und die Referenz

Formel	Ein Ausdruck, der den Inhalt des Elements berechnet
Nachkommastellen	Mit wie vielen Nachkommastellen der Ergebnis des Ausdrucks angezeigt wird, wenn es nummerisch ist
BeimAnklicken	Ein Makro das ausgeführt wird, wenn der Anwender auf das Steuerelement klickt (nur Referenz)

Spezielle Eigenschaften für die Auswahlgruppe und das Kombinationsfeld

Werteliste	Geben Sie hier pro Zeile die Beschriftung für eines der Auswahlfelder an.
WertelistenTyp	Legt fest, ob die möglichen Eingaben auf die Werte in der Werteliste beschränkt sind oder nicht (nur Kombinationsfeld).

Spezielle Eigenschaften für das Bild

ExportZulassen	Der Anwender kann das Bild auf seiner Festplatte speichern.

Spezielle Eigenschaften für die Linie

Linienart	Wählen Sie die Darstellung der Linie
Richtung	Legt fest, ob es eine horizontale oder vertikale Linie ist.

Spezielle Eigenschaften für den Zeitgeber

Aktion	Ein Makro, das in regelmäßigen Abständen vom Zeitgeber ausgeführt wird
Aktiviert	Legt fest, ob der Zeitgeber das Makro aufruft
Intervall	Anzahl der Sekunden zwischen zwei Aufrufen des Makros

Spezielle Eigenschaften für das Memo und den MemoEditor

ExportZulassen	Der Anwender kann das Bild auf seiner Festplatte speichern.
Formatierung	Der Anwender kann Textteile im Memo in einer anderen Schrift eingeben.
Text	Die Beschriftung für den Schalter, der den Memo-Editor öffnet (nur MemoEditor)

3.5.5 Die Schriftart eines Steuerlements ändern

Sie können pro Formular bis zu zwölf verschiedene Schriftarten verwenden, die vordefinierte Namen wie Standard, Überschrift1, Überschrift2 usw. haben.

So weisen Sie einem Steuerelement eine andere Schriftart zu

1. Selektieren Sie das Steuerelement mit Linksklick.
2. Klicken Sie im Eigenschaftsfenster auf den Schalter mit den drei Pünktchen in der Zeile Schriftart.
3. Wählen Sie aus der Liste die gewünschte Schriftart aus.
4. Falls Sie die Einstellungen für die gewünschte Schriftart noch ändern möchten, klicken Sie auf den Schalter Ändern und stellen die Schrifteigenschaften ein.
5. Bestätigen Sie mit Ok.

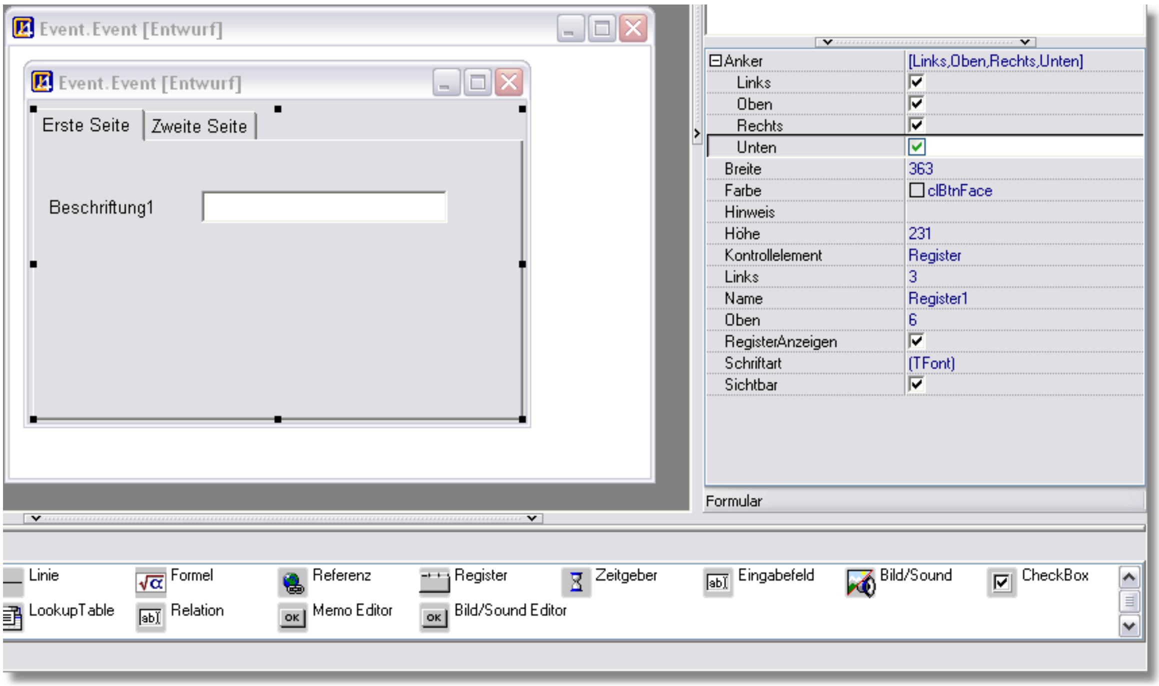
3.5.6 Mehrseitige Formulare erstellen

In *TurboDB Studio* erstellen Sie mehrseitige Formular durch Einfügen eines Register-Elements mit mehreren Seiten, auf welche dann die eigentlichen Steuerelement platziert werden.

Das Register besteht aus einem Rahmen und den einzelnen Seiten. Wenn Sie es in der Mitte anklicken, selektieren Sie eine Seite, wenn Sie neben den Laschen klicken das Register als Ganzes. Dementsprechend sehen Sie im Eigenschaftsfenster einmal die Eigenschaften der gerade angewählten einzelnen Seite oder eben die Eigenschaften des Registers als Ganzes, wie zum Beispiel seinen Namen und die die Verankerung auf dem Formular.

Wenn die Laschen nicht angezeigt sind, muss man zum Selektieren des Register-Rahmens sehr genau auf seinen Rand klicken.

Die Register von TurboDB Studio haben nicht nur den Vorteil, dass man sie auf einen Teil der Formulargröße beschränken kann. Sie können auch weitere Register auf Registerseiten legen.



Das Register wurde hier auf die volle Größe des Formulars aufgezogen und durch Klicken rechts neben die Lasche *Zweite Seite* der Register-Rahmen selektiert. Alle Anker-Seiten - Links, Oben, Rechts und Unten - wurden eingeschaltet.

So wandeln Sie ein einseitiges Formular in ein mehrseitiges Formular um

1. Selektieren Sie alle Steuerelemente im Formular durch Anklicken mit gedrückter Umschalt-Taste.
2. Wählen Sie den Befehl *Bearbeiten/Ausschneiden*, um alle Steuerelemente in die Zwischenablage zu verschieben.
3. Ziehen Sie ein Registerelement auf das Formular und bringen Sie es auf die gewünschte Größe (meist gesamte Größe des Formulars).
4. Wenn Sie möchten, dass das Register mit dem Formular "mitwächst", schalten Sie bei den Eigenschaften alle vier Anker-Seiten ein.
5. Selektieren Sie die erste Registerseite, indem Sie mitten auf das Register klicken und wählen Sie *Bearbeiten/Einfügen* aus dem Menü, um die vorherigen Steuerlemente wieder einzufügen.

Jetzt können Sie über das Kontextmenü mit *Neue Seite einfügen*, neue Seiten zum Register hinzufügen.

So löschen Sie eine Registerseite

Selektieren Sie die gewünschte Seite, indem Sie zuerst die Seite mit ihrer Lasche auswählen und dann durch einen Linksklick selektieren.

Wählen Sie *Bearbeiten/Löschen* aus dem Menü.

TurboDB Studio betrachtet das erste Registerelement im Formular als Hauptregister und verwendet dieses für die Befehle *Nächste Seite*/*Vorherige Seite* sowie beim Umschalten zwischen Formular- und tabellarischer Ansicht.

3.6 Dateneingabe kontrollieren

3.6.1 Dateneingabe kontrollieren

Gerade wenn Sie vorhaben, das von Ihnen erstellte Projekt als Anwendung oder im User-Modus an andere weiterzugeben, ist es sinnvoll, die Dateneingabe im Formular so weit wie möglich zu automatisieren und zu überprüfen. Dazu legen Sie im Formulareditor für das jeweilige Datenfeld die nötigen Eingabehilfen oder Eingabekontrollen fest.

3.6.2 Ein Datenfeld vorbelegen

Oft ist es praktisch, wenn Datenfelder bei der Neueingabe nicht leer sind, sondern sinnvolle Vorgaben enthalten. Z. B. ist das aktuelle Datum eine vernünftige Vorbelegung bei der Bestellannahme. Solche Vorgaben werden mit *TurboDB Studio* durch eine Berechnungsvorschrift für das Datenfeld festgelegt, die für jeden neuen Datensatz einmal ausgeführt wird.

So definieren Sie eine Vorgabe für die selektierten Datenfelder im Formulareditor

1. Selektieren Sie das gewünschte Feld
2. Tragen Sie im Eigenschaftsfenster unter Vorgabe eine Formel ein, mit welcher der Startwert ausgerechnet werden kann.

Die Formel kann dabei eine einfache Konstante sein wie z.B. *'Maier'* oder *1* oder *1.1.2004*. Möglich sind aber auch Funktionsaufrufe (zum Beispiel *Today*), Verweise auf andere Feldinhalte oder Aufrufe selbstgeschriebener Prozeduren.

3.6.3 Eingabemöglichkeit einschränken

Bestimmte Informationen in Datenbanken dürfen, einmal eingegeben, nicht mehr geändert werden. Denken Sie z.B. an Rechnungsbeträge, die keinesfalls editiert werden sollten, wenn die Rechnung schon ausgedruckt und verschickt ist. Datenfelder können in *TurboDB Studio* mit der Spezifikation *NurNeueingabe* versehen werden, um eine nachträgliche Änderung zu verhindern. Der Inhalt solcher Felder kann nur während dem ersten Ausfüllen nach Anlegen des Datensatzes geändert werden.

In anderen Fällen möchten Sie z.B. ein Feld nur unter bestimmten Umständen editieren. z.B. ist die Eingabe eines Geburtsnamens nur bei verheirateten Personen nötig. Im Formulareditor können Sie für jedes Feld eine Bedingung definieren, die als Voraussetzung für die Editierbarkeit gilt.

So verhindern Sie nachträgliche Änderungen eines Datenfeldes

1. Selektieren Sie das gewünschte Steuerelement im Formulareditor.
2. Schalten Sie das Markierungsfeld *NurNeueingabe* im Eigenschaftsfenster ein.

So definieren Sie eine Bedingung für die Editierbarkeit

Selektieren Sie das gewünschte Steuerelement im Formulareditor.

Geben Sie für die Eigenschaft Aktivierungsbedingung eine Bedingung für die Editiermöglichkeit ein. Bei einem Feld für den Geburtsnamen in einer Personaldatei wäre das z.B. *Familienstand ist verheiratet*. Wenn die Bedingung erfüllt ist, kann das Datenfeld editiert werden, ansonsten nicht.

Aktivierungsbedingungen sind normale Bedingungsausdrücke, wie sie beispielsweise auch in der Suche mit Bedingung verwendet werden. Aktivierungsbedingungen können auch selbstgeschriebene Prozeduren aus einem erreichbaren Modul aufrufen. Erreichbar sind bei allen Formular-bezogenen Makros prinzipiell das Formularmodul sowie das erste Modul der Tabelle.

3.6.4 Eingabemuster

Mit einem Eingabemuster kann das Format einer Anwender-Eingabe in ein alphaumerisches Datenfeld festgelegt werden. Dabei kommen die folgenden Zeichen zum Einsatz:

Zeichen	Funktion	Beispiel	Wirkung
L	Buchstabe	LL	Ab ist erlaubt, 3U dagegen nicht
l	(Kleines L) Optionaler Buchstabe	Ll	Ab ist erlaubt, A aber auch.
A	Buchstabe oder Ziffer	AAA	B40, 123, Alt sind alle erlaubt
a	Optionaler Buchstabe oder Ziffer	Aaa	B40, 12, A sind alle erlaubt
C	Irgendein Zeichen	ACA	B-3, abc, X,Y sind alle erlaubt
c	Optionales beliebiges Zeichen	AcA	B-3, B3, abc, ac, X,Y, XY sind alle erlaubt.

0	Ziffer	000	123, 100, 009 sind alle erlaubt
9	Optionale Ziffer	999	123, 12, 1 und leere Zeichenkette sind alle erlaubt.
#	Ziffer oder Vorzeichen	#099	+3, -333, 32 sind alle erlaubt.
_	Automatisch eingefügtes Leerzeichen	00_00	Das Leerzeichen zwischen den Zahlengruppen wird automatisch eingefügt.

Alle anderen Zeichen werden literal behandelt.

Einige einfache Beispiele:

00 000 000	ist das Format für eine Bankleitzahl
90.90.9900	für ein vollständiges Datum mit Trennungspunkt
#0999,999	für eine Dezimalzahl mit Komma

3.6.5 Eine Werteliste anlegen

Wertelisten werden für Eingabefelder angelegt, um die Eingabe einerseits zu erleichtern und andererseits zu beschränken. Wenn Sie für ein Eingabefeld eine zusätzliche Werteliste festlegen, kann man in dieses Feld entweder normal eingeben oder aus der Liste der vorgegebenen Werte einen auswählen. Sie können aber auch eine ausschließliche Werteliste festlegen, wodurch eine Eingabe nur noch durch Auswahl aus der Liste möglich ist.

So legen Sie eine Werteliste für das Datenfeld an:

1. Öffnen Sie den Formulareditor für das gewünschte Formular.
2. Ziehen Sie Eingabeelement Kombinationsfeld auf das Formular.
3. Verbinden Sie das Kombinationsfeld über die Eigenschaft *DatenfeldName* mit der gewünschten Tabellenspalte.
4. Tragen Sie unter Werteliste im Eingabefenster die gewünschten Auswahlmöglichkeiten ein.
5. Wählen Sie mit der Eigenschaft *WertelistenTyp*, ob die angegebenen Auswahlmöglichkeiten die einzigen erlaubten Eingaben darstellen (ausschließlich) oder nur eine Hilfe beim Eintippen (zusätzlich).

3.6.6 Eine Nachschlagetabelle definieren

Nachschlagetabellen haben eine ähnliche Funktion wie Wertelisten. Sie erleichtern die Dateneingabe, indem sie eine Liste von möglichen Einträgen anbieten. Während dies bei der Werteliste in einem Kombinationsfenster geschieht, sind die angebotenen Werte in diesem Fall Einträge in einer anderen Datentabelle, eben der Nachschlagetabelle.

So definieren Sie eine Nachschlagetabelle für das Datenfeld:

1. Öffnen Sie den Formulareditor für das gewünschte Formular.
2. Ziehen Sie das Eingabeelement Nachschlagetabelle auf das Formular.
3. Verbinden Sie die Nachschlagetabelle über die Eigenschaft DatenfeldName mit der gewünschten Tabellenspalte.
4. Wählen Sie bei *NachschlageTabelle* und *NachschlageSpalte* die Tabelle und die Spalte aus, von denen der Wert für das Eingabefeld übernommen werden soll.
5. Zusätzlich können Sie noch für die Anzeige der Nachschlagetabelle ein Formular (*NachschlageFormular*), einen Filter (*NachschlageFilter*) und eine Sortierung (*NachschlageSortierung*) angeben. Wenn Sie das tun wird zur Auswahl des Nachschlage-Datensatzes das angegebene Formular verwendet, wobei die Datensätze mit dem Filter als Selektionskriterium eingeschränkt und nach der Sortierung sortiert werden.

3.6.7 Tabellenspalten berechnen

Manchmal dient ein Datenfeld nicht der freien Eingabe von Informationen, sondern sein Inhalt ergibt sich aus dem Inhalt anderer Datenfelder. Ein Beispiel dafür ist die Berechnung des Gesamtpreises aus Menge und Einzelpreis. Sie können eine Formel als Berechnungsvorschrift für eine Tabellenspalte festlegen. Dadurch wird die Spalte zu einer berechneten Spalte, die ihren Inhalt bei Änderungen des Datensatzes entsprechend anpasst.In ein solches Feld können Sie nichts mehr eingeben.

So definieren Sie eine Berechnung für die Tabellenspalte:

1. Öffnen Sie das Datenmodell.
2. Selektieren Sie die gewünschte Tabelle und wählen Sie Tabellenstruktur im Kontextmenü.
3. Tragen Sie bei der gewünschten Spalte im Feld Formel die Berechnungsvorschrift ein, zum Beispiel *Netto-Preis * 0.16*
4. Das Datenfeld ist jetzt nicht mehr editierbar, passt seinen Inhalt jedoch bei jeder Änderung den aktuellen Daten in den anderen Datenfeldern an.

3.6.8 Eingabe überprüfen

Um die Informationen in der Datenbank so zuverlässig wie möglich zu halten, können Eingaben auf ihre Gültigkeit hin überprüft werden. Z.B. kann das Alter eines Angestellten sicher nicht negativ sein und das Datum seiner Anstellung wird nicht vor dem 1.1.1900 liegen. In solchen Fällen können Sie einfach einen Bereich angeben, in dem die Eingabe liegen muss. Möglich ist dies für Zahl-, Datum- und Zeitfelder.

Eine andere Art der Eingabeüberprüfung besteht darin, eine Gültigkeitsbedingung festzulegen. Wenn Sie eine solche Bedingung als Eingabeüberprüfung definieren, kann ein Datenfeld im Formular nur dann verlassen werden, wenn diese Bedingung erfüllt ist.

So schreiben Sie einen Gültigkeitsbereich für das Datenfeld vor:

1. Selektieren Sie das gewünschte Datenfeld im Formular.
2. Tragen Sie im Eigenschaftsfenster unter *MinimalWert* und *MaximalWert* die gewünschte untere und obere Grenze an, zum Beispiel 5 und 18 oder 3.45 und 3.78 oder 1.1.1900 und 31.12.2199

Wenn nun eine Eingabe in das Datenfeld nicht im angegebenen Bereich liegt, erscheint die Meldung Eintrag muss zwischen xxx und yyy liegen. und das Datenfeld kann nicht verlassen werden.

So legen Sie eine Gültigkeitsbedingung für das Datenfeld fest:

1. Selektieren Sie das gewünschte Datenfeld im Formular.
2. Tragen Sie unter *Gültigkeitsbedingung* die gewünschte Bedingung ein, zum Beispiel *Length(Nachname)* > *1* um die Eingabe von Namen mit nur einem Zeichen zu verhindern.
3. Tragen Sie unter *Ungültigkeitsmeldung* den Text ein, der angezeigt werden soll, wenn der Anwender einen ungültigen Wert eingibt.

3.6.9 Auf Ereignisse reagieren

Mit Datenfeldern und dem Formular selbst können Sie über Ereignisse TurboPL-Makros verknüpfen, die immer dann aufgerufen werden, wenn das entsprechende Ereignis eintritt. Diese Ereignisse sind:

Im Formular

BeimÖffnen

Wird einmal aufgerufen, nachdem das Formular geöffnet wurde.

BeimBetreten

Wird jedesmal aufgerufen, wenn ein anderer Datensatz angezeigt wird. Zum Zeitpunkt des Aufrufs ist der neue Datensatz schon aktuell.

BeimVerlassen

Wird beim Wechsel des Datensatzes aufgerufen. Zum Zeitpunkt des Aufrufs ist der alte Datensatz noch aktuell aber noch nicht in die Datenbank

geschrieben. Es können also noch letzte Änderungen vorgenommen werden.

Nach dem Verlassen

Wird beim beim Wechsel des Datensatzes aufgerufen, nachdem dieser in die Datenbank geschrieben wurde aber noch aktuell ist. Hier kann man zum Beispiel Prüfungen durchführen, die auf den abgespeicherten Datenstand aufsetzen.

BeimSchließen

Wird einmal aufgerufen, bevor das Formular geschlossen wird.

Im Datenfeld

BeimBetreten

Wird aufgerufen, wenn das Datenfeld fokussiert (d.h. angeklickt oder mit der Tab-Taste selektiert) wird.

BeimVerlassen

Wird immer aufgerufen, wenn das Datenfeld verlassen wird. Zum Zeitpunkt des Aufrufs ist das Datenfeld aber noch fokussiert.

BeimÄndern

Wird immer aufgerufen, wenn sich der Inhalt des Datenfeldes ändert.

BeimDoppelklick

Wird aufgerufen, wenn der Anwender auf das Datenfeld doppelt klickt.

So legen Sie Aktionen für Ereignisse von Datenfeldern fest:

1. Öffnen Sie den Formulareditor für das gewünschte Formular.
2. Selektieren Sie das gewünschte Datenfeld im Formular.
3. Im Eigenschaftsfenster sind die Ereignisse BeimBetreten, BeimVerlassen, BeimÄndern und BeimDoppelklick aufgeführt. Suchen Sie das geeignete heraus.
4. Klicken Sie doppelt auf den Eintrag für das Ereignis. Es öffnet sich der Moduleditor mit einer neuen Prozedur, die mit dem Ereignis verknüpft ist.
5. Schreiben Sie die Anweisungen für das Ereignis in die Prozedur und übersetzen Sie das Modul.

Wenn das Ereignis nur einen einfachen Aufruf umfasst, können Sie es auch direkt in das Feld des Eigenschaftsfensters eintragen, zum Beispiel Message('BeimBetreten'). Sie können den Aufruf dann allerdings nicht debuggen.

Beispiel:

Ein häufiger Anwendungsfall für Ereignisse ist, wenn nach dem Ändern eines Datenfeldes andere Datenfelder angepasst werden sollen. Das folgende ist dem Formular einer Applikation zum Ausfüllen von Scheckformularen entnommen. Es formatiert die Eingabe einer Zahl im Feld Betrag zu einer ausgeschriebenen Zahl und trägt sie in das Feld Betrag_in_Buchstaben ein:

```
Betrag_in_Buchstaben := DigitStr(Betrag); Refresh
```

Sie wird immer dann aufgerufen, wenn der Betrag sich ändert, also im Ereignis BeimVerlassen des Feldes Betrag.

3.6.9.1 BeimVerlassen

Auslöser

Wird aufgerufen, wenn der Datensatz geändert wurde und in die Tabelle geschrieben werden soll. Zum Zeitpunkt des Aufrufs ist er aber noch nicht abgeschickt, d.h. in der Datenbanktabelle ist noch die vorherige Version des Datensatzes bzw. kein Datensatz (bei Neueingabe).

Einsatz

- Prüfungen, die vor dem Schreiben des Datensatzes laufen sollen.
- Vergleiche des neuen mit dem vorherigen Datensatz.
- Änderungen am Datensatz, bevor er in die Tabelle geschrieben wird.

Beispiel

Das Formular hat ein Kalenderelement, in das das Datum der letzten Änderung eingetragen wird.

```
procedure ÄnderungsDatumEintragen
  Änderungsdatum.Text := $heute
endproc;
```

Siehe auch

NachDemVerlassen

3.6.9.2 NachDemVerlassen

Auslöser

Wird aufgerufen, wenn eine Datensatz geändert und wieder in die Tabelle geschrieben wurde. Zum Zeitpunkt des Aufrufs ist der geänderte Datensatz der aktuelle Datensatz dieser Tabelle.

Einsatz

In *NachDemVerlassen*, werden Prüfungen und Änderungen vorgenommen, die auf dem Datenbestand in der Datenbanktabelle basieren. Zum Beispiel:

- Prüfungen, die von der Anzahl an Datensätzen mit einer bestimmten Bedingung in der Tabelle abhängen
- Übernehmen von Werten aus dem gerade geschriebenen Datensatz in eine andere Tabelle

Beispiel

Die folgende Prozedur wird in *NachDemVerlassen* aufgerufen, um den Warenbestand zu überprüfen und gegebenenfalls eine Nachbestellung anzuregen. Sie bezieht sich auf eine Tabelle VERKÄUFE, in der alle Warenabgaben eingetragen werden. Die Tabelle VERKÄUFE hat ein Koppelfeld namens *Artikel* zur Tabelle ARTIKEL.

```
procedure BestandPrüfen
  ..Den verkauften Artikel in der Artikeltabelle suchen
  vardef RecN: Integer;
  RecN := FindRec(ARTIKEL, Str(VERKÄUFE.Artikel), 'Artikel.inr', 1);
  ReadRec(ARTIKEL, RecN);
  ..Den Bestand um die Anzahl der verkauften Artikel reduzieren
  ModifyRec(ARTIKEL);
  ARTIKEL.Bestand := ARTIKEL.Bestand - VERKÄUFE.Anzahl;
  PostRec(ARTIKEL);
  .. Falls der Bestand unter die Grenze gefallen ist, Meldung machen
  if ARTIKEL.Bestand < ARTIKEL.MinimalerBestand
    Message('Diesen Artikel sollten Sie jetzt nachbestellen.');
  end;
endproc;
```

Siehe auch

BeimVerlassen

3.7 Berichte gestalten

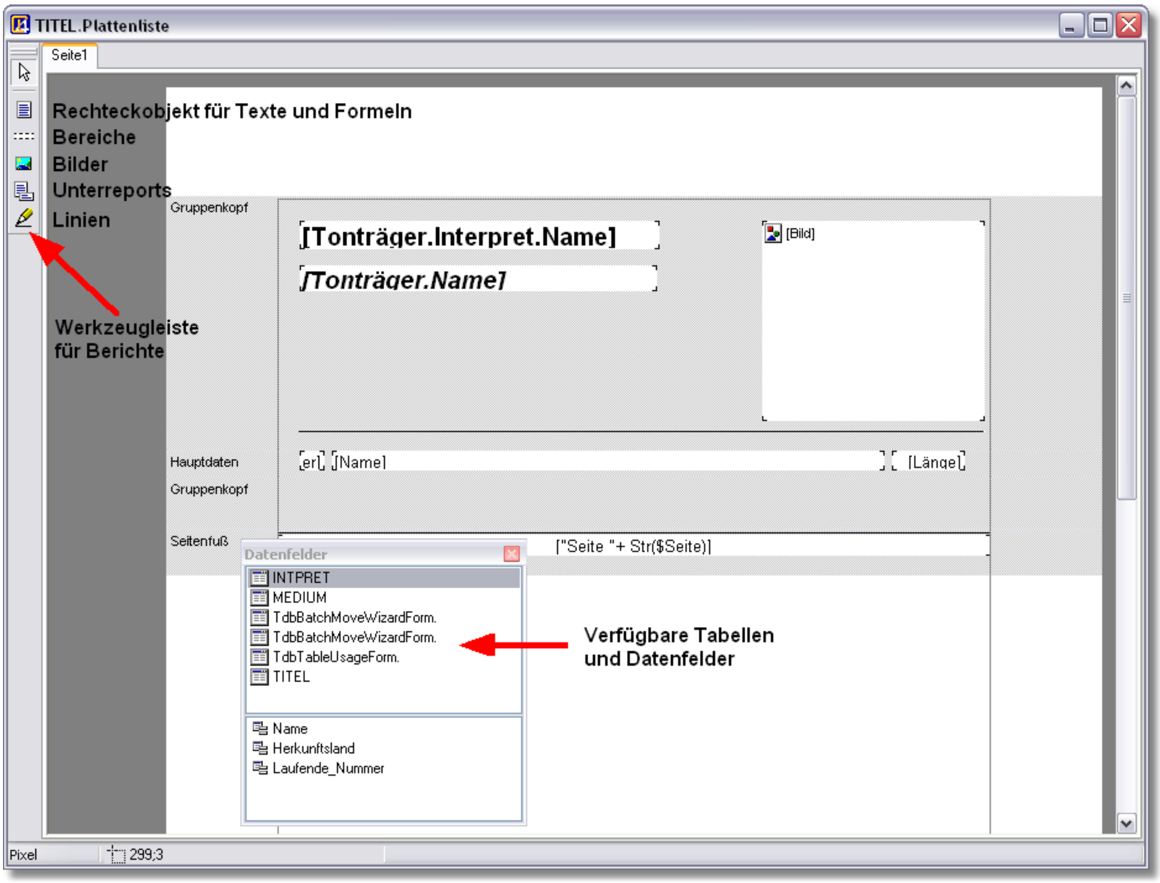
3.7.1 Berichte gestalten

TurboDB Studio bietet Ihnen zwei verschiedene Möglichkeiten, mit Ihren Daten Ausdrucke zu erstellen. Berichte können Sie visuell editieren und die Position und Größe von Ausgabefeldern frei auf einer Seite verschieben. Datenbankjobs werden degegen textuell erstellt und ähneln eher einer Skriptsprache. Tatsächlich sind Datenbankjobs TurboPL-Programmen sehr ähnlich und können die selben Befehle verwenden wie diese. In diesem Kapitel können Sie sich mit der Gestaltung von Berichten vertraut machen, Datenbankjobs werden im nächsten Kapitel besprochen.

Für die grafische Gestaltung von Berichten integriert TurboDB Studio das bekannte Produkt FastReports(R). Die grundsätzliche Arbeitsweise geht so.

So gestalten Sie einen Bericht

1. Rechtsklicken Sie im Projektfenster auf die gewünschte Tabelle und wählen Sie *Neu/Bericht* im Kontextmenü. Der Berichteditor für den neuen Bericht öffnet sich.
2. Fügen Sie einen neuen Bereich in den Bericht ein, indem Sie den Schalter für Bereich in der Werkzeugleiste anklicken und dann die Berichtseite. Es gibt eine ganze Reihe von verschiedenen Bereichen, die sich beim Ausdruck verschieden verhalten.
3. Stellen Sie im Eigenschaftsfenster die gewünschten Werte für den Bereich ein
4. Fügen Sie Felder in den Bereich ein, indem Sie im Fenster Datenfelder zuerst die gewünschte Tabelle anklicken und dann das Feld aus dem unteren Teil des Fensters auf den Bereich ziehen.
5. Stellen Sie im Eigenschaftsfenster die gewünschten Werte für Datenfeld ein
6. Klicken Sie auf den Vorschau-Schalter in der Werkzeugleiste, um eine Ansicht des fertigen Berichts zu sehen.



3.7.2 Bericht-Bereiche

Ein Bericht besteht aus unterschiedlichen Bereichen, die bestimmen, wann und wo die darin enthaltenen Felder ausgedruckt werden. Diese Bereiche entsprechen den Bereichen von Datenbankjobs haben jedoch teilweise andere Bezeichnungen.

Für gewöhnlich befinden sich alle Berichtelement in einem Bereich und nicht direkt auf der Berichtseite.

So fügen Sie einen Bereich in den Bericht ein

1. Klicken Sie auf den Schalter für Bereiche in der Werkzeugleiste für Berichte.
2. Klicken Sie an die Stelle im Bericht, wo Sie den neuen Bereich einfügen wollen.
3. Wählen Sie die gewünschte Bereichsart aus der Liste aus.
4. Ziehen Sie den Bereich auf die gewünschte Position und Größe.
5. Stellen Sie die Eigenschaften des Bereichs im Eigenschaftsfenster ein.



Beim Einfügen eines Bereichs hat man eine große Auswahl an verschiedenen Bereichsarten.

Die Berichts-Bereiche

Report Titel	Wird einmal zu Beginn des Berichts gedruckt (entspricht ungefähr dem Prolog im Datenbankjob)
Report Zusammenfassung	Wird einmal am Ende des Berichts gedruckt (entspricht ungefähr dem Epilog im Datenbankjob)
Seitenkopf	Wird auf jeder Seite oben gedruckt (entspricht dem Kopf im Datenbankjob)
Seitenfuß	Wird auf jeder Seite unten gedruckt (entspricht dem Fuß im Datenbankjob)
Hauptkopf	Wird vor dem ersten Hauptdatensatz gedruckt
Hauptdaten	Wird für jeden Datensatz der Haupttabelle gedruckt (entspricht dem Datenbereich im Datenbankjob)
Hauptfuß	Wird nach dem letzten Hauptdatensatz gedruckt
Detailkopf	Wird vor dem ersten Detail-Datensatz zum Hauptdatensatz gedruckt
Detaildaten	Wird für alle mit dem aktuellen Hauptdatensatz verknüpften Datensätze gedruckt
Detailfuß	Wird nach dem letzten Detail-Datensatz zum Hauptdatensatz gedruckt
Unterdetail Kopf	Wird vor dem ersten Unterdetail-Datensatz zum Hauptdatensatz gedruckt
Unterdetail Daten	Wird für alle mit dem Unterdetail-Datensatz verknüpften Datensätze gedruckt
Unterdetail Fuß	Wird nach dem letzten Unterdetail-Datensatz zum Hauptdatensatz gedruckt

Überlagerung	Wird auf jeder Seite als Hintergrund gedruckt
Spaltenkopf	Wird zu Beginn jeder Spalte gedruckt
Spaltenfuß	Wird am Ende jeder Spalte gedruckt
Gruppenkopf	Wird zum Beginn einer neuen Gruppe gedruckt. (Entspricht dem Gruppenkopf im Datenbankjob.)
Gruppenfuß	Wird am Ende einer Gruppe gedruckt. (Entspricht dem Gruppenfuß im Datenbankjob.)

3.7.3 Bericht-Elemente

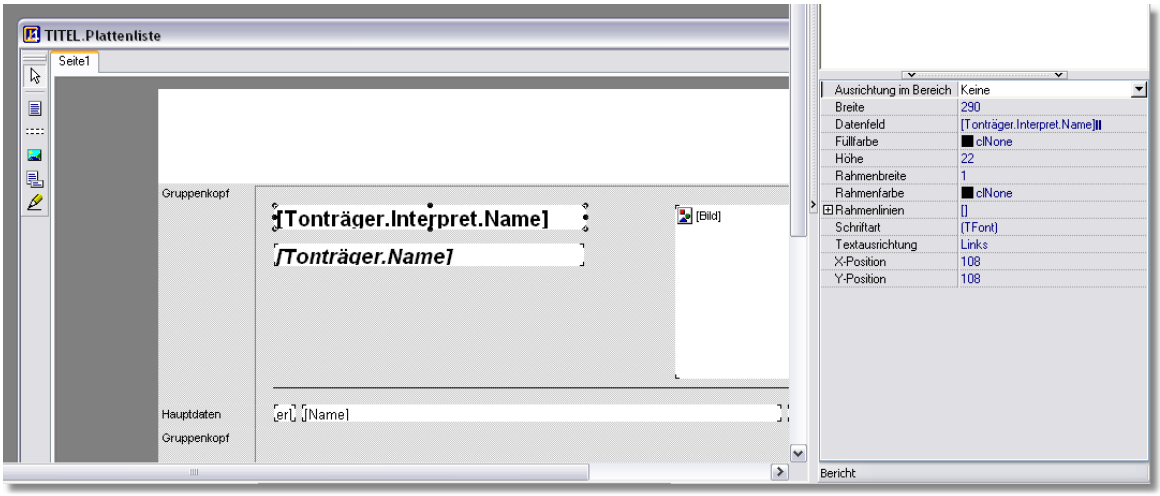
Die eigentlichen Daten werden in Bericht-Elementen angezeigt. Davon gibt es vier verschiedene:

Die Bericht-Elemente

Rechteckobjekt	Textfeld, das entweder mit Daten oder dem Wert einer Formel gefüllt wird.
Bild	Zum Beispiel für das Logo in einem Briefkopf, aber auch für Bilder aus der Datenbank.
Unterreport	Ein Bericht im Bericht, der sich auf den aktuellen Datensatz im Hauptbericht bezieht
Linie	Zum Abgrenzen von Berichtbereichen

So fügen Sie eine Überschrift in den Bericht ein

1. Klicken Sie in der Werkzeugleiste für Berichte das Rechteckobjekt an und legen Sie auf einen Bereich. (Überschriften sind meistens in Titel-, Kopf- und Fußbereichen zu finden.)
2. Doppelklicken Sie auf das Rechteckobjekt, um den Ausdruck-Editor anzuzeigen.
3. Tragen Sie unter Formel eine String-Konstante (in einfachen Anführungszeichen) ein, zum Beispiel 'Überschrift' und schließen Sie den Formel-Assistenten mit Ok.
4. Stellen Sie im Eigenschaftsfenster die Schriftart, die Rahmenart und die Ausrichtung für das Rechteckobjekt ein.

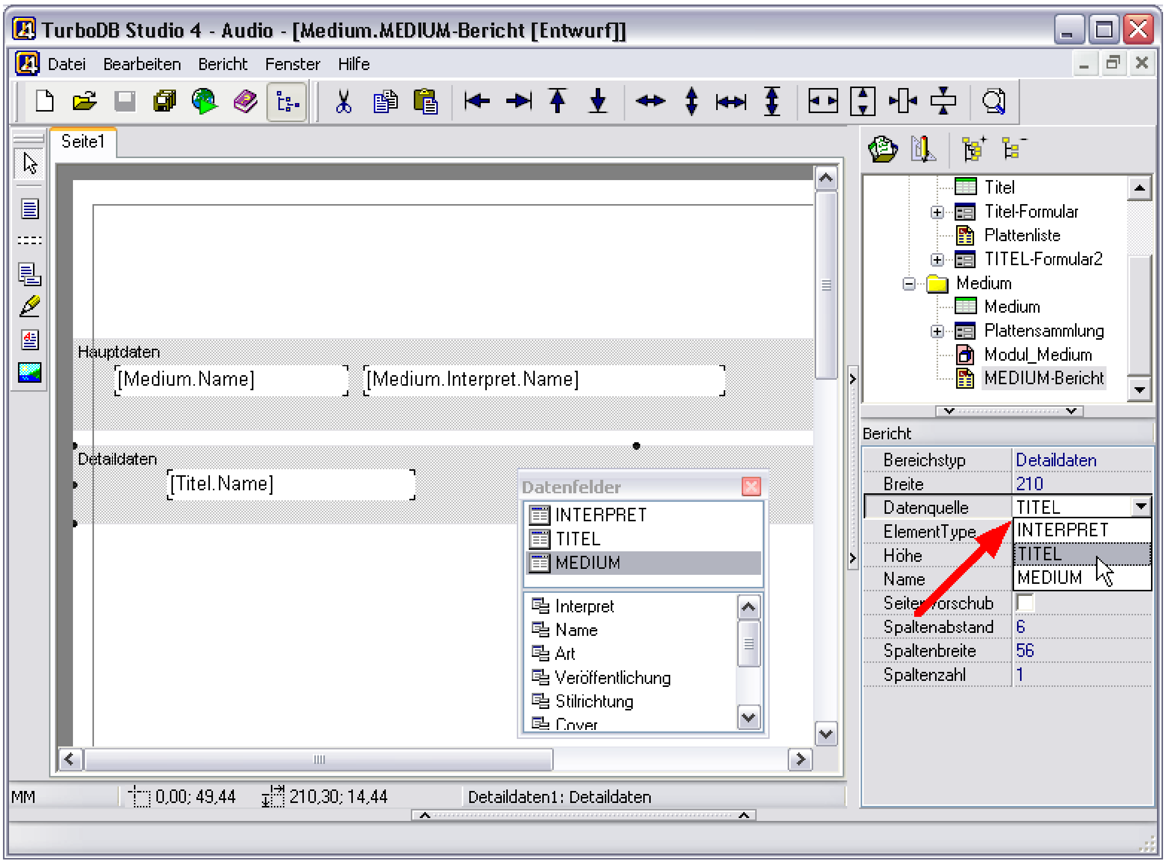


Die Eigenschaften des ausgewählten Bericht-Elements können im Eigenschaftsfenster geändert werden.

3.7.4 Berichte über mehrere Tabellen

Wenn im Bericht Daten aus mehreren Tabelle vorkommen sollen, müssen Sie mit Detail- und Unterdetail-Bändern arbeiten. Eine automatische Kombination der Datensätze wie im Datenbankjob und auch im Berichteditor von Visual Data Publisher unterstützt der Bericht-Editor nicht.

Haupt- und Detaildatenbereiche verfügen über die Eigenschaft Datenquelle. Hier wählen Sie aus, über welche Tabelle dieser Berichtbereich gehen soll.



Um zum Beispiel einen Bericht über Tonträger (Medium) und deren Titel zu drucken, fügen Sie einen Hauptdaten-Bereich für Medium ein und einen Detaildaten-Bereich für Titel.

3.8 Datenbankjobs erstellen

3.8.1 Datenbankjobs erstellen

Datenbankjobs stellen eine Möglichkeit dar, mit ausschließlich textueller Definition Dokumente mit Datenbank-Daten zu drucken. Sie können Datenbankjobs als einen Mittelweg zwischen Programmierung und visuellen Berichteditor betrachten und tatsächlich ist die Syntax von Datenbankjobs in TurboPL eingebettet.

Gegenüber Berichten haben Datenbankjobs sowohl Vor- als auch Nachteile:

- Datenbankjobs haben eine etwas längere Einarbeitungszeit als Berichte.
- Text-orientierte Reports lassen sich mit Datenbankjobs erheblich schneller erstellen als mit Berichten.
- Datenbankjobs werden in reinem Textformat gespeichert und lassen sich deshalb mit jedem beliebigen Texteditor bearbeiten.
- Datenbankjob erlauben die direkte Einbindung beliebiger Programmierelemente aus TurboPL.
- Datenbankjobs erstellen die Grunddatenmenge automatisch aus den Ausgabeformaten und Verknüpfungen im Projekt.

Damit Sie eine erste Vorstellung bekommen, wie ein Datenbankjob aussieht, betrachten wir ein sehr einfaches Beispiel aus dem Demo-Projekt KFZ für die Tabelle KFZ:

```
Datenblatt für $Bezeichnung
```

Dies ist ein vollständiger Datenbankjob, der pro Fahrzeug in der KFZ-Tabelle eine Seite mit dem Text Datenblatt für gefolgt von der Fahrzeugbezeichnung ausdruckt. Das Dollarzeichen macht hier deutlich, dass nicht das Wort Bezeichnung sondern der Inhalt der Tabellenspalte Bezeichnung gemeint ist.

Spaltennamen können mit verschiedenen Formatierungsangaben wie Breite, Anzahl

Dezimalstellen, links- und rechtsbündig versehen werden. Dadurch ist die Zuordnung des Dollarzeichens aber nicht mehr eindeutig und Sie müssen Klammern um das Ausgabeformat machen:

```
Modelljahr $(Modelljahr:10)
```

druckt zuerst den Text Modelljahr und dann den Inhalt der Tabellenspalte Modelljahr in einem Feld der Breite zehn. Die Einheit für alle Größenangaben in Datenbankjobs ist per Voreinstellung die durchschnittliche Größe eines Buchstabens. Die Feldbreite ist im obigen Fall also so, dass die Buchstaben n oder p ungefähr zehnmal darin Platz haben.

Jeder Bericht benötigt unterschiedliche Bereiche für Gesamtüberschriften, Seitenüberschriften, Fußzeilen und so weiter. In Datenbankjobs werden diese durch Kommandos angegeben, denen zur Unterscheidung von Text ein Punkt am Anfang der Zeile vorangestellt wird:

```
.prolog ..Das ist der Bereich für die Gesamtüberschrift
Liste der Fahrzeuge
.daten
Datenblatt für $Bezeichnung

Modelljahr $(ModellJahr:10)
```

Unterschiedliche Schriftarten müssen im Prolog durch das Kommando font definiert werden, um dann bei der Ausgabe mit dem Befehl s*etfont* aktiviert zu werden:

```
.prolog ..Das ist der Bereich für die Gesamtüberschrift
.font A16fu = Arial, 16, f, u
.font A10 = Arial, 10
.setfont A16fu
Liste der Fahrzeuge
.setfont A10
.daten
Datenblatt für $Bezeichnung

Modelljahr $(ModellJahr:10)
```

Sie sehen hier auch dass zwei Punkte einen Kommentar bis zum Ende der Zeile einleiten.

Die Seitenränder werden mit den Steuerbefehlen *PO*, *MT*, *MR* und *MB* eingestellt. Steuerbefehle werden wie Kommandos durch einen Punkt am Zeilenanfang eingeleitet, sie bestehen aber aus nur zwei Buchstaben gefolgt von einer Zahl und können kombiniert werden:

```
.prolog ..Das ist der Bereich für die Gesamtüberschrift
.PO 10, MT 5, MR 10, MB 5 ..Stellt je zehn Zeichen für linken und rechten
sowie 10 Zeilen für oberen und unteren Rand ein
.font A16fu = Arial, 16, f, u
.font A10 = Arial, 10
.setfont A16fu
Liste der Fahrzeuge
.setfont A10
.daten
Datenblatt für $Bezeichnung

Modelljahr $(ModellJahr:10)
```

Für viele Zwecke ist der automatische Seitenumbruch nach jedem Datensatz nicht geeignet. Beginnen Sie den Datenbankjob mit dem Kommando report, um dies zu vermeiden:

```
.report
.prolog ..Das ist der Bereich für die Gesamtüberschrift
.PO 10, MT 5, MR 10, MB 5 ..Stellt je zehn Zeichen für linken und rechten
sowie 10 Zeilen für oberen und unteren Rand ein
.font A16fu = Arial, 16, f, u
.font A10 = Arial, 10
.setfont A16fu
Liste der Fahrzeuge
.setfont A10
.daten
Datenblatt für $Bezeichnung

Modelljahr $(ModellJahr:10)
```

Siehe auch

Referenz Datenbankjobs, Beispiel-Projekt *Datenbankjobs*

3.8.2 Datenbankjobs über mehrere Tabellen

Datenbankjobs berechnen die verwendete Grunddatenmenge selbst nach folgenden Regeln:

1. Die Tabelle, zu welcher der Datenbankjob gehört, gilt als Primärtabelle.
2. Alle im Datenbankjob verwendeten Tabellen, die direkt oder indirekt eine Verknüpfung mit der Primärtabelle haben werden über die entsprechenden inner joins mit aufgenommen. Als Verknüpfungen zählen Koppel- und Relationsfelder sowie die im Datenmodell eingetragenen expliziten Verknüpfungen.
3. Falls weitere Tabellen im Datenbankjob verwendet werden, werden diese über ein vollständiges Kreuzprodukt mit aufgenommen.

Das Relations-Kommando

In seltenen Fällen ist diese automatische Verknüpfung nicht das, was man für einen Report benötigt. Sie können dann die Grunddatenmenge durch das Kommando *relation* selbst definieren. Hier können Sie z.B. Beziehungen zwischen Tabellen festlegen, die während des Datenbankjobs beachtet werden sollen. Das folgenden Beispiel verknüpft diejenigen Fahrzeuge mit dem Kunden, deren Modelljahr mit seinem Geburtsdatum übereinstimmt:

```
.relation KFZ[Modelljahr] = KUNDEN[(Jahr(Geboren))]
```

Mit Hilfe von virtuellen Tabellen sind Verknüpfungen einer Tabelle mit sich selbst möglich. Mit dem RELATION-Kommando kann man eine schon offene Tabelle unter einem anderen Namen ein zweites Mal öffnen:

```
.relation KUNDEN2 = KUNDEN
```

Dies enstpricht dem Erzeugen eines zweiten benannten Cursors für die Tabelle KUNDEN.

Mit einer Inklusion legen Sie schließlich fest, von welchen Tabellen auch dann Datensätze ausgegeben werden sollen, wenn keine verknüpften Datensätze in anderen Tabellen existieren. Eine Liste der Fahrzeuge mit den Käufern z.B. würde nur diejenigen Fahrzeuge ausdrucken, die auch wirklich verkauft worden sind. Mit dem Eintrag

```
.relation KUNDEN
```

kämen alle zur Ausgabe. Damit entspricht die Inklusion einem outer join auf der Seite der entsprechenden Tabelle.

Die drei beschriebenen Kommandos können Sie auch beliebig kombinieren. Sie müssen dann aber alle in einem einzigen Relations-Kommando unterbringen, da jedes RELATION alle vorhergehenden überschreibt:

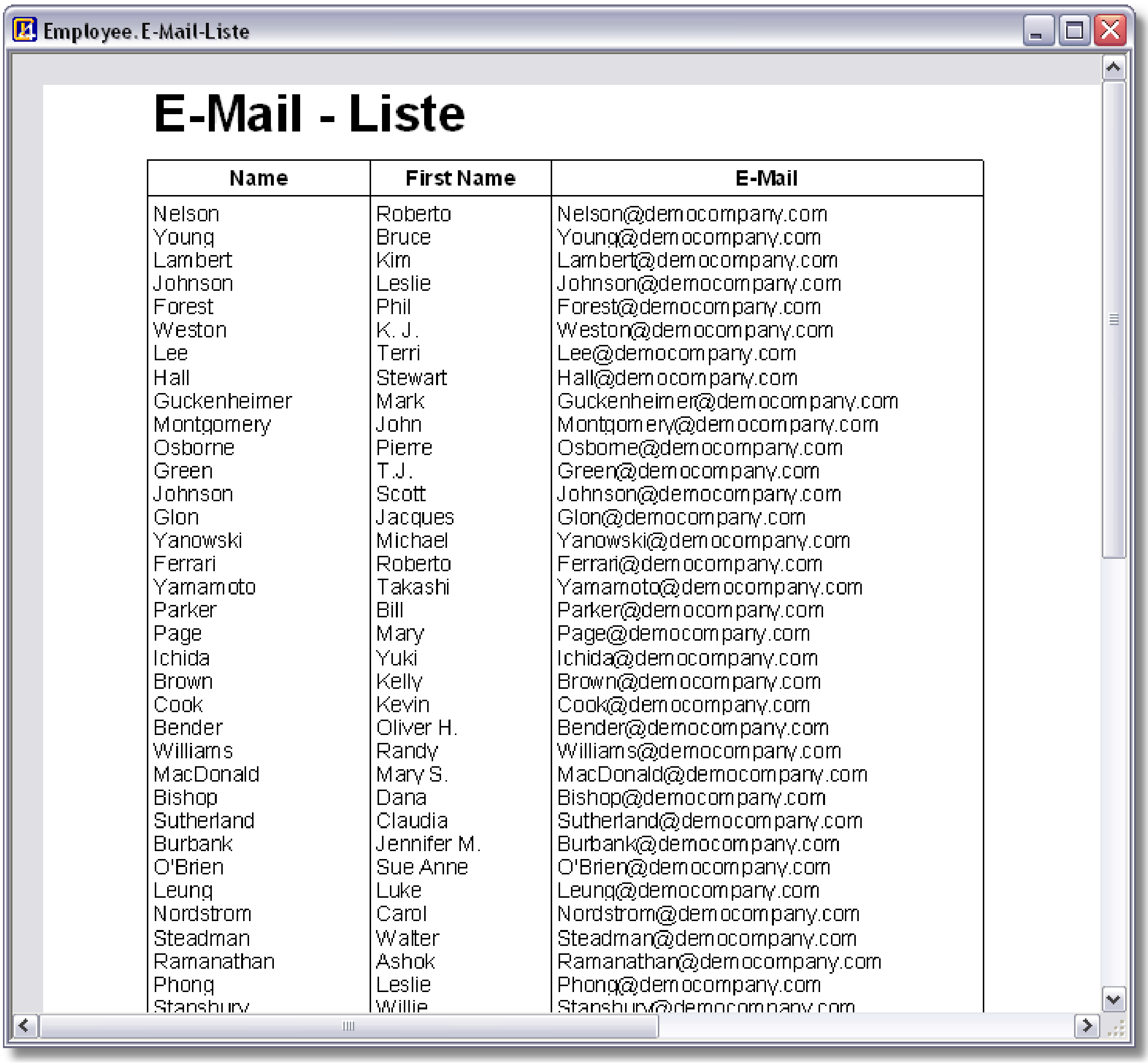
```
.relation KUNDEN2 = KUNDEN, KUNDEN, KFZ[Modelljahr] = KUNDEN[(Jahr(Geboren))]
```

Siehe auch

relation

3.8.3 Reports mit Linien gestalten

In Reports können Sie Gitterlinien erzeugen, um die Zeilen und Spalten besser voneinander abzuheben. Dazu verwenden Sie die Steuerbefehle VL, VX, EVL sowie HL. Der Schnappschuss zeigt das Beispiel aus dem Projekt Datenbankjobs.



Name	First Name	E-Mail
Nelson	Roberto	Nelson@democompany.com
Young	Bruce	Young@democompany.com
Lambert	Kim	Lambert@democompany.com
Johnson	Leslie	Johnson@democompany.com
Forest	Phil	Forest@democompany.com
Weston	K. J.	Weston@democompany.com
Lee	Terri	Lee@democompany.com
Hall	Stewart	Hall@democompany.com
Guckenheimer	Mark	Guckenheimer@democompany.com
Montgomery	John	Montgomery@democompany.com
Osborne	Pierre	Osborne@democompany.com
Green	T.J.	Green@democompany.com
Johnson	Scott	Johnson@democompany.com
Glon	Jacques	Glon@democompany.com
Yanowski	Michael	Yanowski@democompany.com
Ferrari	Roberto	Ferrari@democompany.com
Yamamoto	Takashi	Yamamoto@democompany.com
Parker	Bill	Parker@democompany.com
Page	Mary	Page@democompany.com
Ichida	Yuki	Ichida@democompany.com
Brown	Kelly	Brown@democompany.com
Cook	Kevin	Cook@democompany.com
Bender	Oliver H.	Bender@democompany.com
Williams	Randy	Williams@democompany.com
MacDonald	Mary S.	MacDonald@democompany.com
Bishop	Dana	Bishop@democompany.com
Sutherland	Claudia	Sutherland@democompany.com
Burbank	Jennifer M.	Burbank@democompany.com
O'Brien	Sue Anne	O'Brien@democompany.com
Leung	Luke	Leung@democompany.com
Nordstrom	Carol	Nordstrom@democompany.com
Steadman	Walter	Steadman@democompany.com
Ramanathan	Ashok	Ramanathan@democompany.com
Phong	Leslie	Phong@democompany.com
Stansbury	Willie	Stansbury@democompany.com

Der Steuerbefehl VL definiert den Startpunkt einer senkrechten Linie und wird meist mehrfach angewendet:

```
.VL 10, VL 50, VL 20, VL
```

definiert Startpunkte für vier Linien, wobei die erste am linken Seitenrand positioniert wird, die zweite zehn Einheiten weiter rechts, die dritte wiederum 50 Einheiten weiter rechts und die vierte 20 Einheiten weiter rechts. Der angegebene Platz ist der reine für Daten zur Verfügung stehende Raum. Zusätzlich wird auch noch ein Abstand zwischen den Linien und dem Text freigelassen. Die gesamte Tabelle ist also breiter als 80 Einheiten, nämlich 80 Einheiten plus vier Linien-Umgebungen. Per Vorgabe wird links und rechts von einer senkrechten Linie jeweils eine halbe Zeichenbreite freigelassen, ober- und unterhalb einer senkrechten Linie jeweils eine viertel Zeichenhöhe. Diese Vorgaben können allerdings mit den Befehlen *DX* und *DY* geändert werden. Der Steuerbefehl *VL* wird normalerweise im Kopf eines Reports eingesetzt.

Nach dieser Definition erzeugen Sie mit:

```
.HL
```

eine waagrechte Linie von der ersten zur letzten vertikalen Linie. Im Schnappschuss wurden die beiden oberen horizontalen Linien mit je einem solchen Befehl im Kopf des Datenbankjobs erzeugt.

Die Linien werden gezogen, sobald der Steuerbefehl EVL ausgeführt wird. Dieser Befehl zeichnet

alle vertikalen Linien bis zur letzten gezogenen horizontalen Linie. Somit schaffen Sie mit

```
.HL
.EVL
```

im Fußbereich des Datenbankjobs einen sauberen Abschluss der Tabelle.

Siehe auch

Beispielprojekt *Datenbankjobs*

3.8.4 Datensätze gruppieren

Um die ausgegebenen Daten mit Zwischenüberschriften oder Zwischen-Auswertungen zu versehen, können Sie die Datensätze mit dem Kommando *.group* oder *.gruppe* gruppieren. Zu diesem Kommando gehört ein Ausdruck, die Gruppendefinition. Jedesmal wenn dieser Ausdruck sich ändert, werden die Anweisungen im Gruppenbereich bearbeitet. So führt der Datenbankjob

```
.report
.prolog
.sortBy Hersteller
.group Hersteller

$Hersteller
-------------
.daten
$Modelljahr $Bezeichnung
```

zur Ausgabe der Hersteller als Zwischenüberschrift. Beachten Sie, dass per Vorgabe die Ausgabe des Gruppenbereichs nach dem Wechsel stattfindet. D.h der obige Datenbankjob druckt keine Überschrift vor der ersten Gruppe und dafür eine Überschrift nach der letzten Gruppe. Dies ist normalerweise nicht die gewünschte Variante und kann mit dem Steuerbefehl gp geändert werden.

Wenn Sie im Gruppenbereich auf den Wert der Gruppendefinition zugreifen wollen können Sie dies mit der System-Variablen *G_Neu* erreichen, sie liefert den Wert nach dem Gruppenwechsel. Benötigen Sie dagegen in der Gruppe den Wert vor dem Gruppenwechsel, definieren Sie selbst eine Variable *G_alt*. Sobald diese vorhanden ist, wird sie im Datenbankjob automatisch korrekt belegt.

Somit können wir den obigen Datenbankjob folgendermaßen erweitern:

```
.report
.prolog
.sortBy Hersteller
.gp 0
.var G_alt = Hersteller
.group Hersteller

----------------------------------
Ende der Fahrzeuge von $Hersteller

Fahrzeuge von $Hersteller
-------------------------
.daten
$Modelljahr $Bezeichnung
```

Um eine Zwischenauswertung der Daten im Gruppenbereich vorzunehmen greifen Sie auf die statistischen Funktionen ZSUM und ZCOUNT zu:

```
.report
.prolog
.sortBy Hersteller
.gp 0
.var G_alt = Hersteller
.group Hersteller

---------------------------------------------------
Ende der Fahrzeuge von $Hersteller ($(ZCOUNT(Bezeichnung)) Einträge)

Fahrzeuge von $Hersteller
-------------------------
.daten
```

```
$Modelljahr $Bezeichnung
```

In dieser Variante erhalten Sie am Schluss jeder Gruppe noch die zusätzliche Information, wie viele Fahrzeuge von diesem Hersteller aufgelistet wurden.

3.8.5 Schriftarten in Datenbankjobs

Schriftarten in Datenbankjobs werden durch die Merkmale Schriftartenname, Schriftgröße sowie die Auszeichnungen fett, kursiv und unterstrichen definiert. Eine auf diese Weise festgelegte Schriftart erhält einen Namen, unter dem sie im Folgenden einsetzbar ist. Die Definition der Schriftarten erfolgt im Prolog:

```
.report
.prolog
.font Üs1=Arial,20,f
.font Üs2=Arial,16,f
.font Std=Arial,10
```

Mit diesen Zeilen werden drei Schriftarten namens Üs1 (Überschrift 1), Üs2 und Std (Standard definiert). Sie basieren alle auf der Windows Schriftart Arial und sind 20, 16 und 10 Punkt groß. Die beiden Überschriften werden außerdem fett dargestellt. Die Auszeichnungen werden wie folgt abgekürzt:

f (fett) oder b (bold)
k (kursiv) oder i (italic)
u (unterstrichen oder underline)

Wenn eine Schriftart mehrer Auszeichnung kombiniert, können diese einfach hintereinandergeschrieben werden:

```
.font Üs=Times New Roman,12,ku
```

Definiert eine 12 Punkt Times New Roman, die sowohl kursiv als auch unterstrichen ist.

Um diese Schriftart tatsächlich zu verwenden, gibt es das Kommando *setfont*:

```
.daten
.setfont Üs2
$Nachname, $Vorname
.setfont Std
$Geburtsdatum, $Geburtsort
```

Damit werden die Felder Nachname und Vorname groß und fett ausgedruckt, während Geburtsdatum und -ort normal gedruckt werden.

Auch die Umschaltung der Schriftart innerhalb der Zeile ist möglich mit dem @-Zeichen:

```
$(@Üs2 Nachname @Std Geburtsdatum)
```

wird der Nachname mit Überschrift 2 und das Geburtsdatum in der Standard-Schrift ausgegeben.

Soll nur die Auszeichnung in der Zeile geändert werden, müssen Sie keine eigene Schriftart definieren. Die Funktionen Bold und Italic übernehmen die Umschaltung:

```
$(Bold(1) Nachname Bold(0) Vorname)
```

druckt den Nachnamen fett und den Vornamen mager in der gerade eingestellten Schriftart und Schriftgröße. Italic tut das entsprechende mit der Auszeichnung kursiv.

3.9 Daten mit anderen Programmen austauschen

Wenn Sie Daten von TurboDB Studio in andere Anwendungen übernehmen möchten, oder Daten aus anderen Anwendungen in TurboDB Studio nutzen wollen, haben Sie folgende Möglichkeiten:

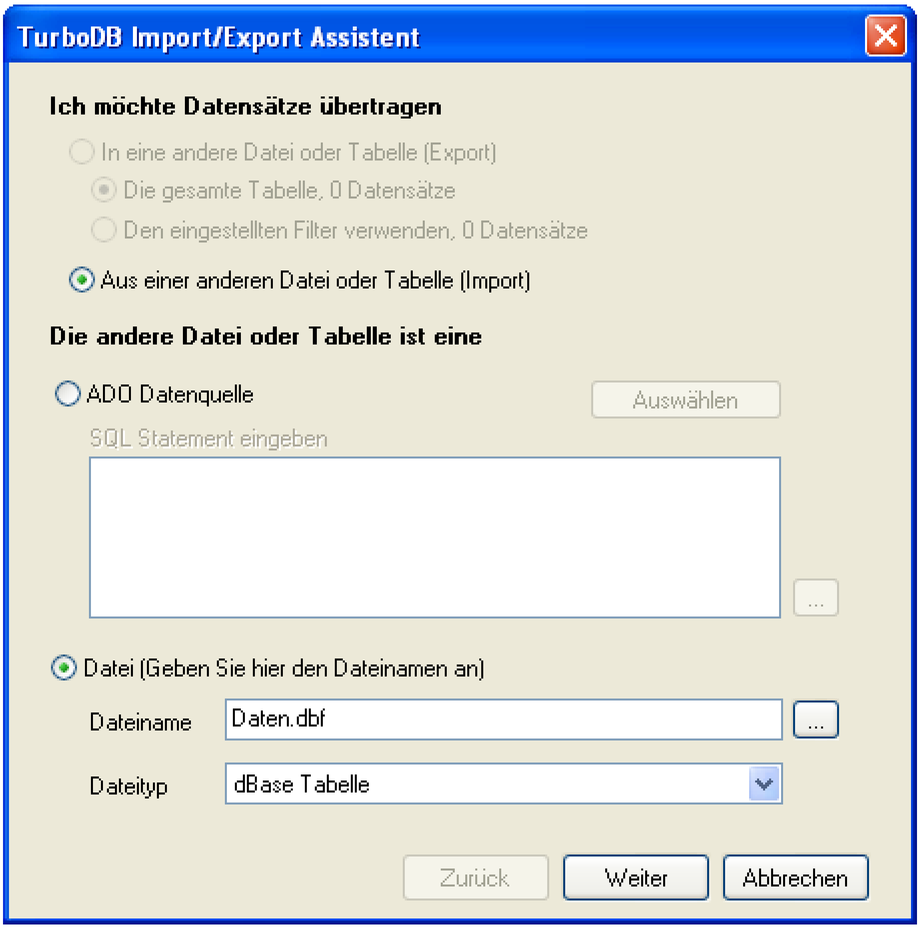
- Daten importieren, die in einem Text-, dBase- oder TurboDB-Format vorliegen
- Daten aus einer anderen Datenbank importieren
- Daten in ein Text-, dBase- oder TurboDB-Format exportieren
- Berichte und Datenbankjobs als Text- oder HTML-Dateien speichern

Außerdem können Sie auch per Programm andere Anwendungen über Automation steuern. Dieses Thema wird in "Ole-Automation verwenden" angesprochen.

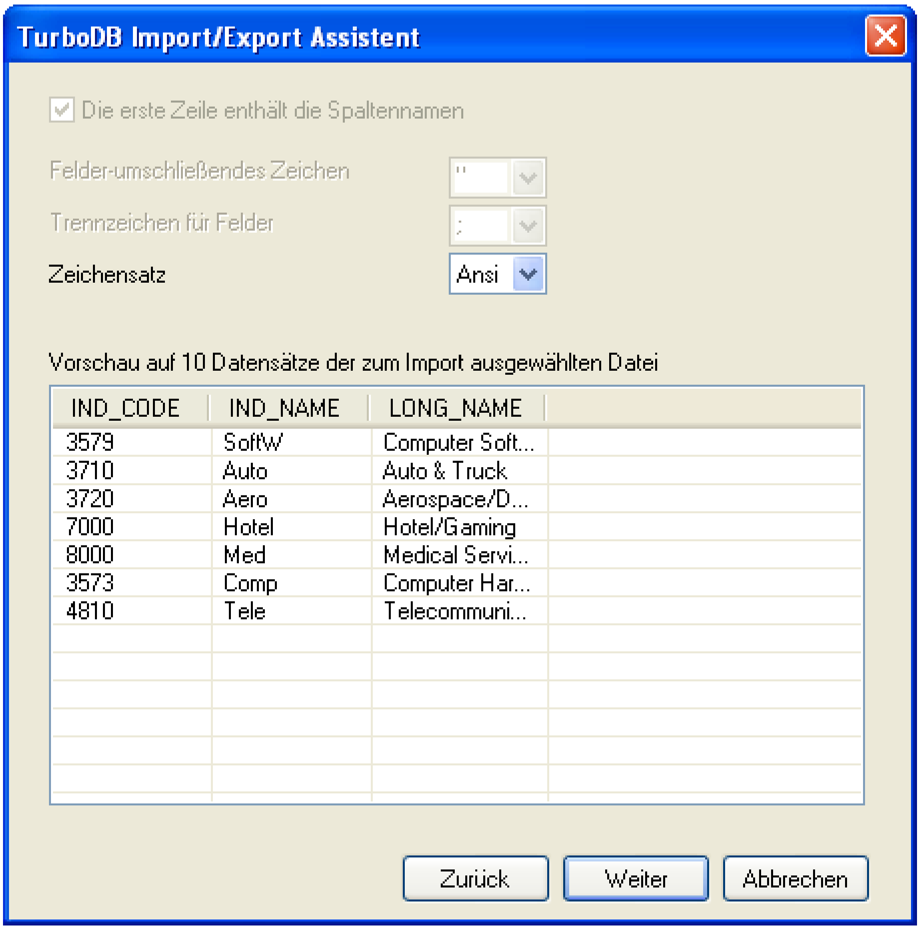
3.9.1 Daten aus Datei importieren

TurboDB Studio kann Daten aus Textdateien, dBase-Dateien und anderen TurboDB-Dateien importieren Voraussetzung ist, dass die Tabelle, in die importiert werden soll schon existiert, und dass die Spalten in der zu importierenden Tabelle zumindest teilweise den Spalten in der Zieltabelle entsprechen.

1. Öffnen Sie das Standard-Tabellenfenster der Tabelle, in die Sie importieren wollen.
2. Wählen Sie *Bearbeiten/Datensätze* importieren im Menü, um den Import/Export-Assistenten zu öffnen.



3. Markieren Sie die Felder *Aus einer anderen Datei oder Tabelle (Import)* und *Datei*.
4. Stellen Sie unter Dateityp den gewünschten Typ ein und klicken Sie dann auch den Schalter mit drei Punkten, um die Datei auszuwählen.
5. Klicken Sie nun auf Ok und dann auf Weiter; Sie sehen nun die Vorschauseite, wo sie den Aufbau der zu importierenden Daten überprüfen können.



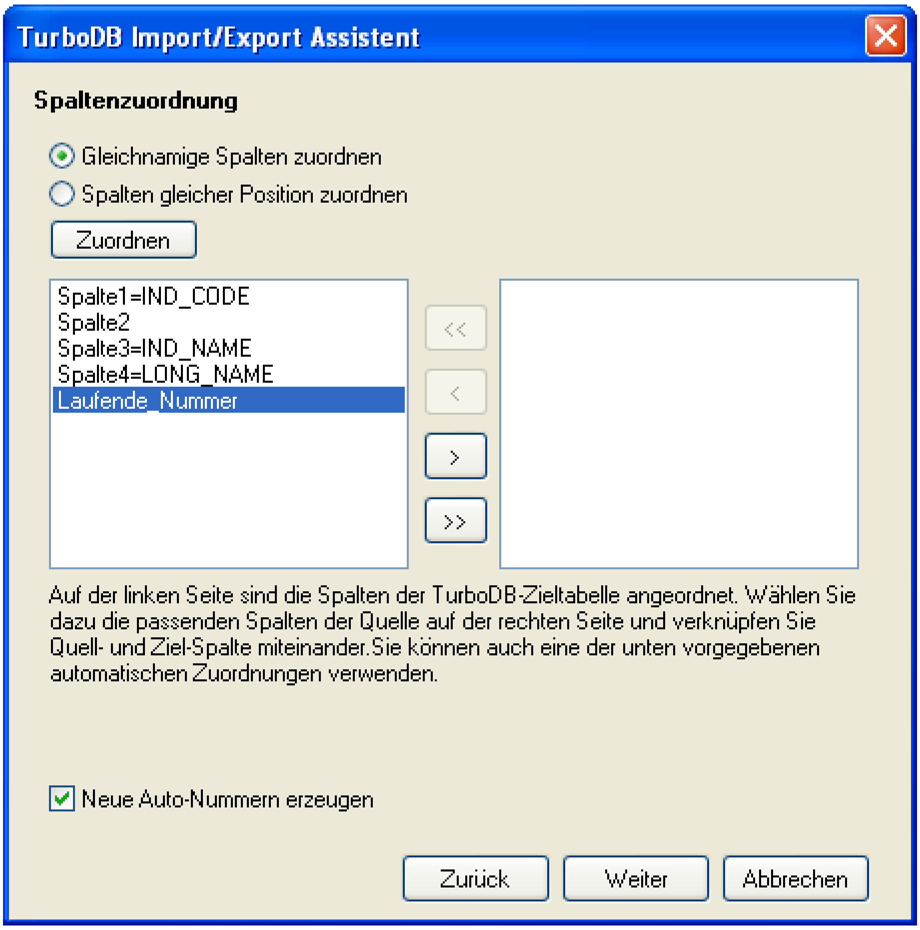
6. Wenn die Vorschau Ihren Erwartungen entspricht, klicken Sie auf Weiter.
7. Hier können Sie den Spalten der Zieltabelle die gewünschten Spalten der Quelltabelle zuweisen. Dazu haben Sie drei Möglichkeiten:

a) Wenn die zueinander gehörigen Spalten der Zieltabelle und der Quelltabelle gleich heißen, können Sie sie mit der entsprechenden Option automatisch zuweisen.

b) Wenn Sie die Spalten einfach von links nach rechts zuordnen wollen, d.h. die erste Spalte der Quelltabelle an die erste Spalte der Zieltabelle usw, ebenso.

c) Ansonsten können Sie mit Hilfe der Schalter zwischen den beiden Spaltenlisten die Spalten einzeln manuell zweisen.

Im folgenden Screenshot werden die Daten aus der Spalte IND_CODE der Quelltabelle in die Spalte Spalte1 der Zieltabelle übernommen, aus der Spalte IND_NAME in die Spalte Spalte3 und so weiter.



8. Aktivieren Sie die Option Auto-Nummern neu vergeben, falls Sie möchten, dass die importierten Datensätze neue Auto-Nummern enthalten. Wenn diese Option nicht aktiviert ist, übernimmt TurboDB Studio die Werte der Auto-Nummern aus der zu importierenden Tabelle. Drücken Sie dann auf Weiter, um die Seite mit den Import-Details anzuzeigen.
9. Wählen Sie nun die Aktion auf der Quelltabelle. Beim Import können Sie nicht nur einfach die Datensätze der Quelltabelle in die Zieltabelle übernehmen, Sie können auch:

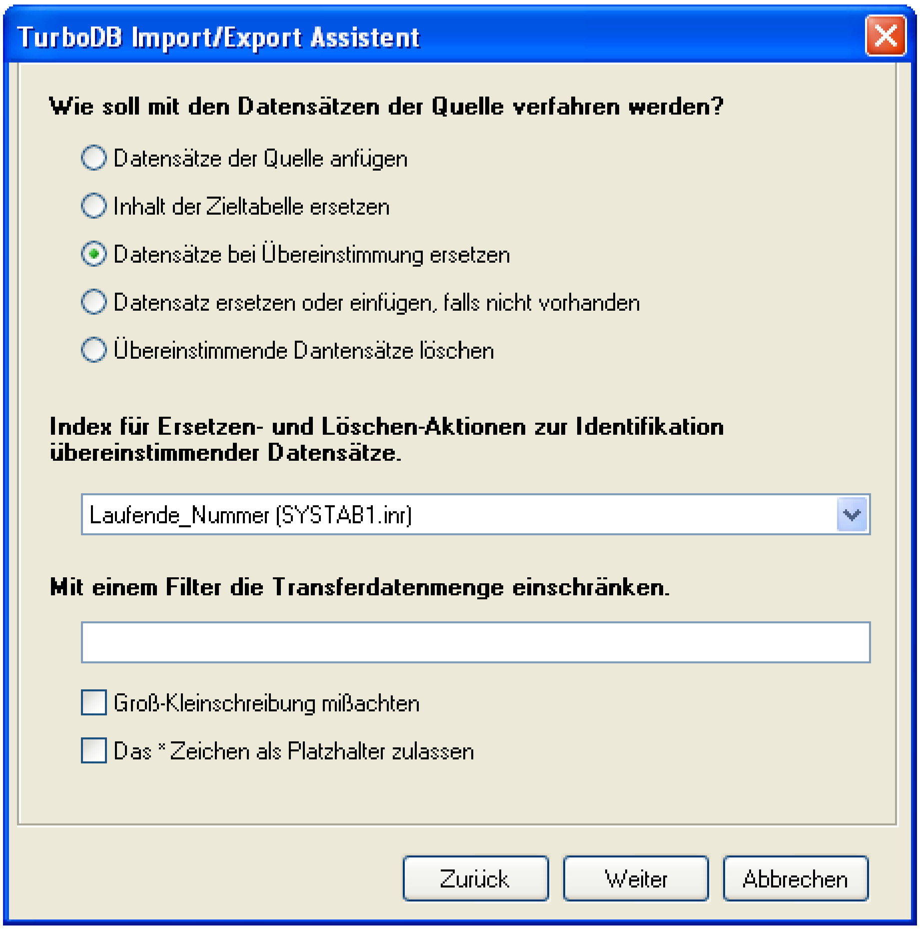
a) Nur diejenigen Datensätze übernehmen, die in der Quelltabelle noch nicht vorhanden sind

b) Die Zieltabelle vor der Übernahme der Datensätze löschen

c) Datensätze in der Zieltabelle durch entsprechende Datensätze der Quelltabelle ersetzen. Damit können Sie neuere Versionen der Einträge in die Tabelle zurückspielen.

d) Datensätze ersetzen, wenn sie einem Datensatz der Quelltabelle entsprechen und anfügen, falls nicht. Diese Option aktualisiert die Zieltabelle mit geänderten und angefügten Datensätzen aus der Quelltabelle.

e) Datensätze, zu denen es einen passenden Datensatz in der Quelltabelle gibt, löschen. Diese Option können Sie dazu nutzen, eine Tabelle in zwei oder mehr Tabellen aufzuspalten.



10. Bei all diesen Import-Arten außer Option b) muss TurboDB feststellen, ob ein Datensatz aus der Quelltabelle einem Datensatz aus der Zieltabelle entspricht. Dies geschieht über eine Feldkombination, für die ein Index vorhanden ist. Dieser Index wird im Normalfall ein eindeutiger Index sein und wahrscheinlich dem Primärschlüssel entsprechen. Wenn die Werte der Feldkombination in der Quelltabelle denen in der Zieltabelle entsprechen, dann werden die Datensätze als zueinander gehörig betrachtet. Angenommen, beide Tabellen haben eine Spalte Kundennummer und in der Zieltabelle ist beim Datensatz mit der Kundennummer 1287 das Geburtsdatum als 1.1.1980 eingetragen und in der Quelltabelle als 1.1.1970. Dann verhält sich der Import über den Index der Kundennummer bei den verschiedenen Import-Arten wie folgt:

a) Der Datensatz in der Zieltabelle bleibt erhalten.

b) Der Datensatz in der Zieltabelle wird gelöscht und der aus der Quelltabelle eingetragen.

c) Der Datensatz in der Zieltabelle wird durch den der Quelltabelle ersetzt.

d) Wie c)

e) Der Datensatz in der Zieltabelle wird gelöscht.

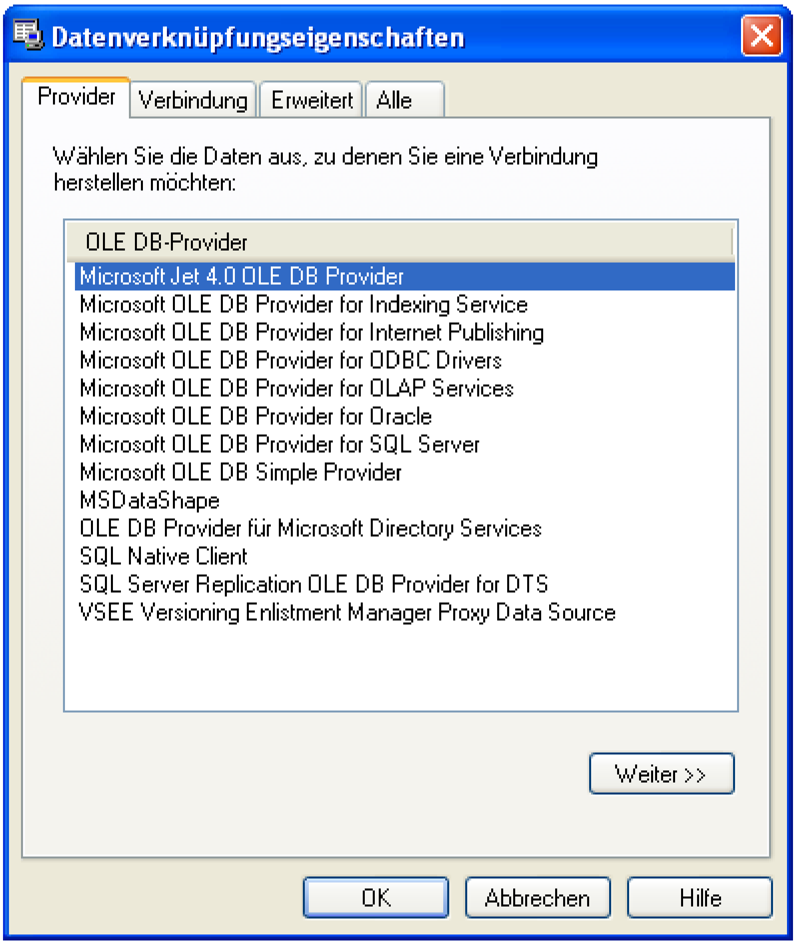
11. Zusätzlich können Sie noch einen Filter definieren, der die Datensätze für diese Import-Aktionen einschränkt. Ein Datensatz, der dem Filter nicht entspricht, wird gar nicht weiterbearbeitet, d.h. ein entsprechender Datensatz der Zieltabelle bleibt in jedem Fall erhalten. Wenn Sie das Feld Groß-/Kleinschreibung missachten markieren, wird beim Vergleich von Zeichenketten auch mit =, <, > nicht zwischen großen und kleinen Buchstaben unterschieden. Die zweite Option bedeutet, dass beispielsweise der Filter *Name = 'Schm*'* für *Schmid*, *Schmidt*, *Schmidt*, *Schmalz* usw. zutrifft.

12. Wenn Sie nun auf Weiter klicken und dann auf Transfer, startet der Import und schließt hoffentlich mit einer Erfolgsmeldung ab.

3.9.2 Daten aus Datenbank importieren

TurboDB Studio kann Daten aus allen Datenbanken importieren, die ADO unterstützen. Der Import wird mit dem Import/Export-Assistenten vorgenommen, und ist teilweise identisch mit dem *Import aus Datei*. Im folgenden werden nur die unterschiedlichen Teile des Vorgehens ausführlich beschrieben und bei den anderen auf den *Import aus Datei* verwiesen.

1. Öffnen Sie ein Tabellenfenster für die gewünschte Zieltabelle und wählen Sie Bearbeiten/Datensätze importieren...
2. Markieren Sie Aus einer anderen Datei oder Tabelle und ADO Datenquelle. Klicken Sie auf Auswählen, um eine ADO-Datenquelle zu definieren.
3. Als erstes bestimmen Sie einen Treiber für die ADO-Datenquelle und klicken auf Weiter.

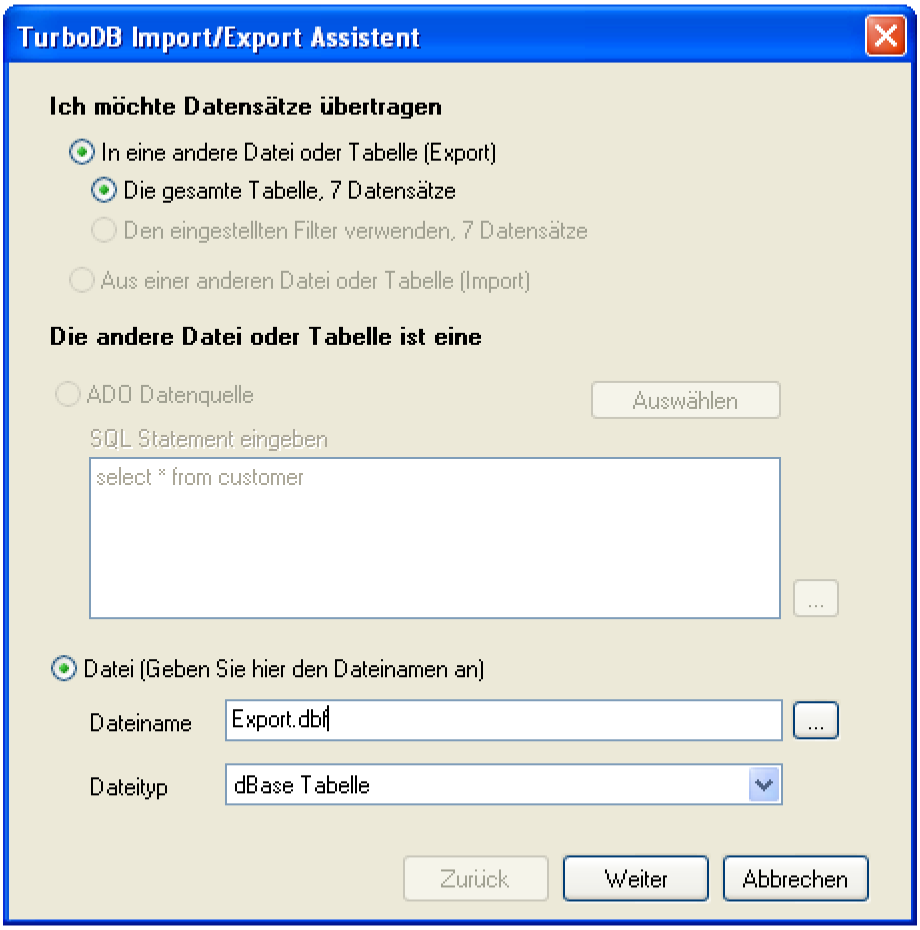


4. Dann wählen Sie die Datenbank aus. Dieser Schritt hängt davon ab, welchen Treiber Sie im vorherigen Schritt gewählt haben. Evtl. benötigen Sie auch einen Benutzernamen und ein Passwort für den Zugriff auf die Quell-Datenbank. Klicken Sie auf Verbindung testen und dann auf OK, wenn die Verbindung hergestellt werden konnte.
5. Das weitere Vorgehen ist wie bei *Import aus Datei*, allerdings wird keine Vorschau angezeigt.

3.9.3 Daten exportieren

TurboDB Studio kann die Tabellendaten in Text-Dateien, dBase-Dateien oder TurboDB-Dateien exportieren. Dabei wird grundsätzlich eine neue Tabelle angelegt, die (soweit möglich) dasselbe Schema wie die Quelltabelle aufweist, und mit allen Datensätzen gefüllt.

1. Öffnen Sie das Tabellenfenster für die Quelltabelle und wählen Sie Bearbeiten/Datensätze exportieren... im Menü, um das Dialogfeld für den Export anzuzeigen.



2. Wählen Sie unter Dateityp das gewünschte Datenformat für den Export.
3. Tragen Sie bei *Dateiname* den Namen der Zieldatei ein. TurboDB Studio wird diese Datei neu erzeugen.
4. Klicken Sie auf *Weiter* und dann auf *Transfer*, um den Export zu starten.

3.9.4 Auswertungen in Datei ablegen

In den vorherigen Abschnitten haben Sie erfahren, wie Sie druckreife Auswertungen Ihrer Daten erstellen können. Dabei haben Sie bereits die Möglichkeiten zur Ausgabe auf einen Drucker oder in die Druckvorschau kennengelernt. Zusätzlich können Ihre Auswertungen auch in eine Datei ausgegeben werden, was z.B. für folgende Fälle in Frage kommt:

- Die Auswertung dient einer anderen Anwendung als Datenquelle (vgl. Export).
- Es sollen HTML-Seiten erzeugt werden.
- Der Drucker ist am Rechner nicht angeschlossen

Für diese Fälle bietet es sich an das gewünschte Format sowie den Namen des Ausgabedatei im Druckdialog direkt anzugeben. Dieses ist jedoch nur möglich, wenn der Bericht aus dem Projektfenster gestartet wird bzw. der Befehl Drucken die Auswahl gestattet.

Um den Anwender den Umgang mit dem Programm zu erleichtern, ist es üblich, das Ausgabeziel bereits beim Befehl Drucken mit anzugeben. Dieses ist für "normale" Ausdrucke ausreichend, beim Ausdruck in eine Datei muss zusätzlich auch der Name der Zieldatei mit angegeben werden. Zu diesem Zweck existiert der Befehl *SetzeAusgabeDatei*, mit dessen Hilfe nun der passende Dateiname angegeben werden kann.

Ausgabe des Berichtes "Adressliste" als HTML-Datei:

```
procedure ErzeugeWebAdressliste
   SetzeAusgabeDatei("C:\Adressliste.htm", 4))
   Drucken("KUNDEN.Adressliste", 3);
endproc
```

Siehe auch:

Drucken, SetzeAusgabeDatei

3.10 Makros und Programme einsetzen

3.10.1 Makros und Programme verfassen

Makros und Programme werden in der Turbo Programming Language *TurboPL* (ehemals *easy*) abgefasst. Makros sind oft nur einzelne Aufrufe, die in einem Formular gespeichert sind auf bei Eintritt eines Ereignisses, z.B. beim Verlassen eines Feldes aufgerufen werden. Programme bestehen aus mehreren Dateien mit *TurboPL* Quelltext, den Modulen. Programme werden zu P-Code übersetzt und in prg-Dateien abgelegt.

So fügen Sie dem Formular ein Makro hinzu:

1. Wählen Sie das gewünschte Formular im Projektfenster aus klicken Sie im Kontextmenü auf *Entwerfen*.
2. Selektieren Sie im Formulareditor ein Steuerelement des Formulars.
3. Tragen Sie im Eigenschaftsfenster im Feld *BeimVerlassen* den Text: *Message('Beim Verlassen')* ein.
4. Speichern Sie das Formular und öffnen Sie es durch Doppelklick im Projektfenster.
5. Schalten Sie den Bearbeitungsmodus auf Editieren.
6. Ändern Sie den aktuellen Wert des Feldes für das Sie das Makro eingetragen haben und verlassen Sie das Feld.
7. *TurboDB Studio* führt das Makro aus und zeigt die Mitteilungsbox mit dem Text *Beim Verlassen* an.

So schreiben Sie ein TurboPL-Programm:

1. Öffnen Sie das Kontextmenü im Projekt und wählen Sie *Neu*, dann *Modul*.
2. Geben Sie einen geeigneten Dateinamen an und klicken Sie auf Ok.
3. Im Moduleditor tragen Sie folgenden Text ein:

```
procedure MeineErsteProzedur
    vardef Name: String;
    if Input('Geben Sie ihren Namen an:', 'Meine erste Prozedur', 0, Name) = 1
       Message('Ihr Name ist ' + Name + '.');
    end;
endproc;
```

4. Wählen Sie *Modul/Übersetzen* im Menü und anschließend *Modul/Prozedur starten*. Sie sehen nun die Liste aller Prozeduren aus dem aktuellen Modul (im Moment nur eine).
5. Wählen Sie eine davon aus und klicken Sie auf Starten.
6. Das Programm zeigt zuerst ein Eingabedialogfeld an, wo Sie ihren Namen eintragen können. Anschließend kommt ein Mitteilungsfeld mit dem Text und ihrem Namen.

Beim Schreiben von Prozeduren im Modul-Editor können Sie auf drei verschiedene Arten Informationen über verfügbare Prozeduren erhalten:

1. Mit der Tastenkombination Strg-Leertaste öffnet sich eine Liste mit den Prozeduren und Variablen, die an dieser Stelle zur Verfügung stehen.
2. Wenn Sie begonnen haben einen Prozeduraufruf einzutippen, können Sie hinter der öffnenden runden Klammer mit der Tastenkombination Strg+Umschalt+Leertaste ein Hilfefenster mit den Parametern für diese Funktion anzeigen.
3. Wenn Sie die Einfügemarke auf einem Schlüsselwort von *TurboPL* positionieren und dann die Taste *F1* drücken, öffnet sich die Online-Hilfe an der Stelle für dieses Schlüsselwort.

3.10.2 Programme debuggen

TurboPL Prozeduren können schrittweise abgearbeitet werden, um den Ablauf des Makros zu untersuchen und evtl. vorhandene Fehler zu finden. Dabei kann man auch den Wert aller gerade sichtbaren Variablen kontrollieren. Es ist möglich, auf bestimmte Programmzeilen Haltepunkte zu setzen, so dass der Ablauf wiederum anhält, wenn diese Zeile erreicht wird.

So debuggen Sie ein Makro:

1. Öffnen Sie die Makro-Liste auf eine dieser beiden Arten:
    - im Projektfenster durch Doppelklick auf das Modul
    - im Datenfenster über den Menüpunkt Ausführen/Makro...
2. Selektieren Sie das gewünschte Makro und klicken Sie auf Debug. Das Makro wird im Debug-Modus gestartet, d.h. es bleibt gleich in der ersten Programmzeile stehen und zeigt diese im Moduleditor an.
3. Nun haben Sie mehrere Möglichkeiten:
    - Führen Sie die aktuelle Programmzeile aus, indem Sie den Schalter für Überspringen anklicken.
    - Falls in der aktuellen Programmzeile ein Prozeduraufruf steht, gehen Sie mit Betreten zur ersten Zeile in dieser Prozedur.
    - Klicken Sie auf Starten, um die Ausführung des Makros ganz normal fortzusetzen. Erst beim nächsten Haltepunkt bleibt es wieder stehen.
    - Wählen Sie Zurücksetzen, wenn Sie die Ausführung des Makros abbrechen wollen.
    - Setzen Sie einen Haltepunkt auf eine andere Programmzeile, indem Sie auf dieser Programmzeile am linken Rand des Moduleditors klicken und Haltepunkt setzen im Kontextmenü auswählen.
    - Untersuchen Sie die lokalen Variablen, im Fenster "Lokale Variablen" ganz unten im TurboDB Studio Rahmenfenster.
    - Untersuchen Sie weitere Variablen und Ausdrücke im Auswerte-Fenster, das Sie mit dem Schalter Auswerten öffnen können.

So werten Sie Variable und Ausdrücke aus:

1. Starten Sie das Debuggen wie oben beschrieben.
2. Während das Programm angehalten ist, öffnen Sie das Auswerte-Fenster mit dem Schalter Auswerten.
3. Tragen Sie im Feld Ausdruck den Namen einer Variablen oder einen Ausdruck wie z.B. sin(5) ein.
4. Klicken Sie auf Auswerten. Im Feld Wert wird der Wert des Ausdrucks angezeigt, falls dieser gültig ist (bei sin(5) also z.B. -0,96). Falls der Ausdruck nicht gültig war, erhalten Sie eine entsprechende Meldung.

3.10.3 Datenfenster steuern

Die meisten Aktionen, die der Anwender interaktiv mit einem Datenfenster ausführen kann, sind auch über *TurboPL*-Makros möglich. Dazu gehören:

- Datenfenster öffnen und schließen mit *ActivateForm*, *OpenForm* und *CloseWnd*
- Sortierung einstellen mit *SetSortOrder*
- Im Datenfenster navigieren mit *ShowRec*
- Datensätze suchen und markieren mit BySelection

und viele mehr. Eine vollständige Übersicht gibt es in der *TurboPL* Referenz unter Benutzerschnittstelle.

Wenn Sie eine Funktion für ein Datenfenster ausführen muss natürlich klar sein, an welches Datenfenster sich diese Funktion richtet. Sie haben dazu zwei Möglichkeiten. Entweder Sie verwenden das gerade aktive Datenfenster oder ein ganz spezielles.

Das gerade aktive Datenfenster wird immer dann benutzt, wenn Sie eine Oberflächenfunktion in

einem Applikationsmodul aufrufen ohne explizit ein Datenfensterobjekt anzugeben.

In allen anderen Fällen wird ein spezielles Formular benutzt:

- Das Makro, das für ein Ereignis aufgerufen wird, wendet sich an das Formular, in dem das Ereignis aufgetreten ist.
- In einem Formularmodul wird das Datenfenster benutzt, das die Prozedur aufgerufen hat, also das Formular zu dem das Modul gehört.
- Wenn dem Aufruf ein Datenfensterobjekt vorausgeht, wird dieses verwendet. Ein Datenfensterobjekt erhalten Sie entweder durch den Aufruf von *FindDataWnd* oder durch den Verweis auf eine eingebettete Tabelle in einem Formularmodul.

Für den dritten Fall hier ein kurzes Beispiel, in dem wir annehmen, dass die folgende Zeile im Formularmodul eines Formulars für die Tabelle KFZ steht, welches eine eingebettete Tabelle für die Tabelle KUNDEN enthält. Die eingebettete Tabelle heißt *KundenTabelle*.

```
KundenTabelle.ShowRec(8)
```

zeigt dann den Datensatz Nummer 8 in der eingebetteten Kundentabelle an.

Würde im selben Formularmodul eine Prozedur die Zeile

```
ShowRec(8)
```

enthalten. Würde in diesem Formular selbst der Datensatz acht angezeigt werden. Dabei spielt es keine Rolle, ob das Formular gerade das aktive ist, oder nicht. Auch wenn es zum Zeitpunkt des Aufrufs völlig verdeckt von anderen Formularen wäre, würde der Aufruf im Formularmodul an genau dieses Formular gehen.

Anders, wenn der selbe Aufruf in einem Applikationsmodul steht. In diesem Fall wird immer das gerade aktive Datenfenster benutzt, egal von wo aus der Aufruf kommt. Der Grund dafür ist, dass *ShowRec(8)* im Applikationsmodul eigentlich ein Aufruf für die Applikation ist, welche dann das aktive Datenfenster sucht und dort den Datensatz anzeigt. *ShowRec(8)* im Formularmodul ist dagegen ein direkter Aufruf ans Formular und landet deshalb immer bei genau diesem.

3.10.4 Modale Datenfenster ausführen

Modale Datenfenster dienen der Eingabe oder Auswahl von Datensätzen, ohne dass der Anwender in dieser Zeit mit anderen Formularen als dem modalen etwas tun kann. Wenn ein modales Datenfenster geöffnet ist, sind alle anderen Datenfenster und auch sonstige Fenster deaktiviert. Anders als bei Dialogen ist aber das Menü und die Schalterleiste für das modale Datenfenster noch bedienbar. In einem modalen Datenfenster kann der Anwender also je nach Variante noch Datensätze wechseln, eingeben, suchen, markieren und so weiter.

Auch wenn Sie noch nie eines in ihren Programm verwendet haben, kennen Sie modale Datenfenster wahrscheinlich aus der normalen Bedienoberfläche. Die Nachfrage bei Koppel- und Relationsfeld-Eingabe, Nachschlagetabellen sowie die bei Ansicht/Verknüpfte Datensätze.. und Bearbeiten/Verknüpfte Datensätze angezeigten Datenfenster sind modal. Modale Datenfenster haben neben dem eigentlichen Formular noch ein sogenanntes Panel mit einem oder mehrere Aktionsschaltern und einem erklärenden Text, dem Kommentar.

Es gibt eine ganze Reihe von TurboPL-Funktionen die modale Datenfenster anzeigen. Sie unterscheiden sich darin, was der Anwender in diesem Datenfenster machen darf und was nicht. Von den meisten Funktionen gibt es eine Einzahl- und eine Mehrzahl-Form wie zum Beispiel *EditRec* und *EditRecs*. Die Mehrzahl-Form erlaubt es, den aktuellen Datensatz zu wechseln, während die Einzahl-Form auf dem vom Programm eingestellten Datensatz verharrt. Dies sind die verfügbaren Funktionen:

ViewRec/ViewRecs	Anwender kann Datensätze betrachten aber nicht ändern.
ChooseRec/ MarkRecs	Anwender kann einen oder mehrere Datensätze auswählen aber nicht ändern.
EditRec/EditRecs	Anwender kann einen oder mehrere Datensätze editieren.
AppendRec/ AppendRecs	Anwender kann Datensätze zur Tabelle hinzufügen, aber nicht ändern.
ViewLinkedRecs	Anwender kann die mit dem aktuellen Datensatz verknüpften Datensätze betrachten.
AppendLinkedRec	Anwender kann neue mit dem aktuellen Datensatz verknüpfte Datenätze

s eintragen.

Die typischen Schritte beim Ausführen eines modalen Datenfensters sind diese:

1. Das Formular für das Datenfenster öffnen.
2. Alle Einstellungen für das Datenfenster vornehmen: Auswahl, Sortierung, aktueller Datensatz, angezeigte Formularseite usw.
3. Die modale Aktion starten.
4. Prüfen, auf welche Weise der Anwender die modale Aktion beendet hat (Bestätigung oder Abbruch) und entsprechend handeln.
5. Das Formular schließen

Weil in diesem Ablauf das Formular erst geöffnet wird, finden sich Prozeduren, die mit modalen Datenfenstern arbeiten für gewöhnlich in einem Applikationsmodul. Die folgende Prozedur erfragt vom Benutzer, welche der offenen Rechnungen gedruckt werden sollen und markiert diese intern, die Bedeutung der Tabellen und Spalten ergibt sich aus den verwendeten Bezeichnern:

```
procedure AuswahlAusOffeneRechnungen: Integer;
   vardef Result: Integer;
   OpenForm('RECHNUNG.Auswahlliste');
   MitBedingung('Status = Offen', 1, 1);
   Sortierung('Rechnungsnummer');
   if MarkRecs('Markieren Sie die Rechnungen, die gedruckt werden sollen')
      ..Die Oberflächemarkierungen zu internen machen
      Sortierung('Markierung');
      Result := 1;
   else
      Result := 0;
   end;
   CloseWnd;
   return Result;
endproc;
```

3.10.5 Dialoge ausführen

Ein Dialog ist ein modal ausgeführtes Fenster, dass solange die Eingabe in andere Fenster unterbindet, bis man ihn mit OK oder Abbrechen beendet hat. Ein Dialog hat gewisse Ähnlichkeiten mit einem modalen Datenfenster, allerdings sind während der Anzeige des Dialogs auch das Menü und die Schalterleiste deaktiviert, die man in einem modalen Datenfenster noch bedienen kann. Ein Dialog wird normalerweise verwendet, um eine oder mehrere Eingaben vom Benutzer zu erfragen, ohne die die Ausführung eines Vorgangs nicht fortgesetzt werden kann.

In TurboDB Studio ist ein Dialog ein Formular, das mit der TurboPL-Funktion ExecDialog modal ausgeführt wird. Dadurch können für Dialoge alle Möglichkeiten von Datenfenstern genützt werden:

- Die eingegebenen Daten werden automatisch in einer Datenbanktabelle gespeichert und deshalb beim nächsten Aufruf automatisch wieder angezeigt.
- Die Ereignisse für das Betreten und Verlassen von Steuerelementen und Formular können auch für den Dialog genutzt werden.
- Gültigkeitsprüfungen und Eingabehilfen stehen auch für Dialoge zur Verfügung.

Weil Dialogdaten meistens keine "normalen" Datenbankdaten sind, sondern meistens nur in einem Exemplar benötigt werden, erscheint die Verwendung einer Tabelle dafür auf den ersten Blick als übertrieben. Allerdings hat es keinen Nachteil, wenn man für solche Daten eine System-Tabelle einrichtet, die oft nur einen einzigen Datensatz enthält, aber unter Umständen viele Felder, nämlich für jedes Steuerelement in einem Dialog eines.

3.10.6 Applikations-Module und Formular-Module

Module, die direkt in einem Ordner hinzugefügt werden, heißen Applikations-Module, weil sie sich auf die Applikation als Ganzes beziehen. Die Oberflächenfunktionen, die hier aufgerufen werden, gehen immer an das gerade aktive Datenfenster.

Formular-Module können sich dagegen streng auf ein Formular beziehen. Sie werden in der

Projektansicht nicht unter dem Ordner für eine Tabelle angezeigt sondern direkt unter dem Formular zu dem sie gehören. Prozeduren aus einem Formular-Modul können nicht aufgerufen werden, wenn das Formular nicht geöffnet ist. Wenn beim Formular die Eigenschaft *FormularbezogeneMakros* aktiviert ist, gehen Oberflächenfunktionen, die im zugehörigen Formular-Modul verwendet werden immer an das Formular, welches die Prozedur aufgerufen hat.

Ein Beispiel:

In einer Anwendung gibt es für die Tabelle T zwei Formulare F1 und F2. Das folgende Makro liegt auf einem Knopf im Formular *F1*:

```
ActivateForm('T.F2');
ShowRec(-3);
```

Wenn der Anwender auf den Schalter klickt, hängt das Ergebnis dieses Makros davon ab, ob die Prozedur in einem Applikations-Modul oder im Formular-Modul des Formulars F1 zu finden ist.

In einem Applikations-Modul wird zuerst die Methode *ActivateForm* der Applikation und dann die Methode *ShowRec* der Applikation aufgerufen. Das *ShowRec* bezieht sich dadurch auf das aktive Formular, also auf F2, das damit den zweiten Datensatz der Tabelle anzeigt.

Das Formular-Modul verhält sich genauso, solange die Option *FormularbezogeneMakros* in der Eigenschaft *Verwendung* nicht gesetzt ist. Falls sie jedoch gesetzt ist, wird zuerst die Methode *ActivateForm* der Applikation (*ActivateForm* gibt es nicht als Formular-Methode) und dann die Methode *ShowRec* des Formulars aufgerufen. Dadurch wirkt das *ShowRec* auf das aufrufende Formular und nicht auf das aktive Formular und der Datensatz bleibt in F2 auf eins und wechselt dafür in F1.

Die Prozeduren in einem Formular-Modul "wissen" in diesem Fall, dass das Formular da ist und können sich deshalb direkt darauf beziehen. Zum Beispiel sind die Namen der Steuerelemente in Formular-Modulen als direkte Namen erlaubt. Wenn das Formular ein Steuerelement namens *Button1* enthält, dann kann eine Prozedur in dem zugehörigen Formular-Modul direkt darauf zugreifen:

```
procedure Proc1InFormModule;
  Button1.Text := "Click me!";
endproc;
```

Dieser direkte Verweis auf das Steuerelement hat gegenüber dem Verweis mit FindControl zwei Vorteile:

1. Die Code-Vervollständigung listet in einem Formular-Modul alle verfügbaren Steuerelemente auf und kennt außerdem den Typ des Steuerelements. Sie erhalten also mehr Unterstützung bei der Programmierung.
2. Wenn Sie ein solches Steuerelement umbenennen oder löschen, wird dies beim nächsten Übersetzen des Moduls bemerkt und nicht erst beim Testen. Dadurch sparen Sie Test-Aufwand.

TurboDB Studio legt für jedes Formular automatisch ein Formular-Modul an. Wenn Sie eine Prozedur zu einem Formular-Ereignis generieren, wird diese Prozedur standardmäßig im zugehörigen Formular-Modul angelegt. Falls die Prozedur zu einem Ereignis nicht im Formular-Modul gefunden wird, wird sie allerdings auch noch im ersten Applikations-Modul der Tabelle gesucht, wie das in den Versionen bis VDP 3 der Fall war.

Prozeduren in Formular-Modulen können von anderen Prozeduren des selben Formular-Moduls aufgerufen werden, nicht jedoch aus anderen Modulen. Das liegt daran, dass die Prozeduren aus einem Formular-Modul zur Ausführung ein Formular des passenden Typs benötigen. Das ist bei Aufrufen aus anderen Modulen nicht möglich.

Das müssen Sie sich merken:

- Eine Formular-Prozedur wie zum Beispiel ShowRec wirkt in einem Formular-bezogenen Makro auf das Formular, auf das sich das Makro bezieht. Ein Applikations-bezogenes Makro wirkt auf das gerade aktive Formular der Applikation.
- Nur in Formular-bezogenen Makros können Sie auf die Steuerelemente des Formulars zugreifen.
- Ein Makro ist Formular-bezogen, wenn es in einem Formular-Modul steht, dessen Formular in der Eigenschaft Verwendung die Option FormularbezogeneMakros aktiviert hat.

Die Beispiel-Anwendung FormControl zeigt den Einsatz eines Formular-Moduls zur Beeinflussung der Steuerelemente.

3.10.7 Oberflächenmarkierungen auswerten und manipulieren

Oberflächenmarkierungen werden oft auch Sternchen (engl. stars) genannt, weil sie in früheren Version von TurboDB Studio so dargestellt wurden. Für Makros ist es oft nützlich, diese abzufragen, zum Beispiel, um die so markierten Datensätze auszudrucken oder auszuwerten. Andererseits müssen Makros die Sternchen setzen und löschen können, um dem Benutzer eine Auswahl (etwa die in einer Suche gefundenen Datensätze) zu visualisieren.

Es gibt zwei grundsätzlich verschiedene Methoden dies zu erreichen. Die erste sind natürlich die Oberflächenfunktionen, die mit Sternchen arbeiten: *MitBedingung*, *DeleteStars*, *DeleteRecords*, *GetStars*, *PutStars*.

Die andere Methode beruht auf der Abbildung zwischen dem Formular-Kontext und dem Datenbank-Kontext beim Start eines Makros oder bei *Attach*. Wenn ein Makro aufgerufen wird, werden die Oberflächenmarkierungen in die Datenbank kopiert und können dann dort als Tabellenmarkierungen (interne Markierungen) mit Funktionen wie *NMarks*, *IsMark* usw. analysiert bzw. mit *SetMark* und *DelMark* gelöscht werden. Bei einem Aufruf von Attach werden diese Tabellenmarkierungen dann wieder ins aktuelle Datenfenster kopiert und als Sternchen angezeigt.

3.10.8 Datensätze importieren und exportieren

Zum Import und Export von Datensätzen wird gewöhnlich ein Programm wie zum Beispiel tdbDataX oder TurboDB Viewer verwendet. Von einem Programm aus, haben Sie folgende Möglichkeiten:

Import/Export mit Dialog

Den Import/Export-Dialog der Oberfläche können Sie mit ImportRecords aufrufen und damit den selben Komfort für den Datenaustausch anbieten, wie TurboDB Studio selbst.

Textdateien

Textdateien können mit den Befehlen *Reset* und *Rewrite* geöffnet werden. Je nach Bedarf können dann die Datensätze z.B. als Textzeilen eingelesen und in die Einzelwerte zerlegt werden. Die Funktionen *StrToReal*, *StrToDateTime* usw. helfen bei der Konvertierung der String-Werte in die korrekten Datenbankwerte. Mit *ReadRec*/*WriteRec* bzw. *Append*/*Replace* werden die Datensätze dann in die Tabelle geschrieben.

TurboDB-Dateien

Hier kann man mit *OpenDB* die Tabelle öffnen und dann mit *SetRecord*, *GetField*/*SetField* oder direkten Zuweisungen die Datensätze von einer Tabelle zur anderen zu transferieren.

Andere Dateien

Wenn ein ODBC-Treiber installiert ist, können Sie die Daten auch über ODBC und die Funktion ImportODBC importieren.

3.10.9 Indexe nutzen

Vorhandene Indexe werden in Makros und Programmen teils implizit und teils explizit genutzt. Implizit werden sie dann genutzt, wenn das System beim Sortieren oder Auswählen von Datensätzen erkennt, dass ein Index Geschwindigkeitsvorteile bringt. Die explizite Nutzung besteht in erster Linie daran, dass man den Zugriff auf eine Tabelle bewusst auf einen Index setzt, um die entsprechende Reihenfolge der Datensätze zu erhalten. Dies geschieht mit der Funktion *Access*. Um zu ermitteln, welcher Index bei einer Tabelle gerade gesetzt ist, kann man die Funktion *IndNo* verwenden. Diese liefert eine Zahl, die man anschließend mit *IndName* oder *IndDef* in einen lesbaren String konvertieren kann.

3.10.10 Mit Datum und Uhrzeit rechnen

Einige der arithmetischen Operatoren können auch auf die Datums- und Uhrzeittypen angewendet werden. Dabei ist immer wichtig, zwischen Zeitpunkten und Zeitspannen zu unterscheiden. TurboPL benutzt für Zeitpunkte die Datentypen *Time*, *Date* und *DateTime*, für Zeitspannen immer Real-Zahlen, die je nach Rechnung eine Anzahl von Tagen oder eine Anzahl von Minuten bedeuten.

Addition

Zu einem Datum oder einem Zeitstempel kann eine Anzahl von Tagen (evt. mit Bruchteil) addiert werden. So liefert die folgende Formel das Datum in einer Woche vom aktuellen Tag an gerechnet:

```
Today + 7
```

Bei Uhrzeiten wird die Zeitspanne in Minuten angegeben. So können Sie zum Beispiel auf die aktuelle Uhrzeit 30 Sekunden dazuaddieren:

```
Now + 0.5
```

Zwei Zeitpunkte kann man dagegen nicht addieren, die Summe etwa aus dem 1.3.2004 und dem 12.10.1978 macht ja genauso wenig Sinn, wie die Summe aus 8:42 Uhr und 22:01 Uhr.

Subtraktion

Wenn man zwei Zeitpunkte voneinander abzieht, erhält man die Zeitspanne dazwischen, zum Beispiel liefert die folgende Formel die Anzahl der Tage, die seit dem Beginn der französischen Revolution verstrichen sind. Der Datentyp des Ergebnisses ist natürlich eine Real-Zahl und kein Datum:

```
Today - 14.7.1789
```

Man kann aber auch eine Zeitspanne von einem Zeitpunkt abziehen und erhält dann wie bei der Addition den entsprechenden neuen Zeitpunkt (als *Time*, Date oder *DateTime*). Das folgende berechnet die Uhrzeit von vor fünf Minuten und ist damit vom Typ *Time*:

```
Now - 5.0
```

Vergleiche

=, <, >, <=, >= sind zwischen Zeitpunkten erlaubt. Dabei kann man Datum mit Datum, Datum mit Zeitstempel und Zeit mit Zeit vergleichen. Eine Zeit kann man jedoch nicht mit einem Datum oder einem Zeitstempel vergleichen. Beispiele:

```
if Today = 1.1.2000
while Now < 12:00:00.000
```

3.10.11 Datum und Uhrzeit formatieren

Wenn Datums- und Zeitwerte in lesbare Ausgabe konvertiert werden, spielt die Formatierung eine wichtige Rolle. Schließlich wollen deutsche Anwender normalerweise das Datum in der Form 23.12.2004 sehen, währende amerikanische Benutzer 12/23/2004 gewohnt sind.

In Formularen und tabellarischen Ansichten hängt die Darstellung von den Einstellungen des Rechners ab. Hier muss sich der Entwickler nicht kümmern.

Anders ist es schon bei Datenbankjobs. Hier werden Daten und Uhrzeiten in einer deutschen Version von *TurboDB Studio* im deutschen Format ausgedruckt, während sie in einer englischen Version im englischen Format gedruckt werden. Dies ist allerdings nur die Voreinstellung und kann per Programm geändert werden. TurboPL bietet die folgenden Datums- und Zeitformate:

- Lokales Format: Das Datum und die Zeit werden entsprechend den Einstellungen es Rechners formatiert
- *TurboDB* Format: Das Datum ist ein deutsches Format mit vierstelliger Jahreszahl und die Zeit im 24-Stunden-Format mit Doppelpunkt als Trenner: 23.12.2004 08:26:35. Die Millisekunden werden mit Punkt abgetrennt.
- Deutsches Format: Wie *TurboDB* Format aber mit zweistelliger Jahreszahl und einem Komma zwischen den Sekunden und Millisekunden. Einstellige Stundenangaben ohne führende 0: 23.12.04 8:26:35.
- Englisches Format: Amerikanische Datums- und Zeitangabe mit einem Punkt für die Millisekunden: 12/23/2004 8:26:35 am
- Internationales Format: Trennung des Datums mit Bindestrichen, Uhrzeit wie bei *TurboDB* Format aber ohne führende 0 bei den Stunden: 2004-12-23 8:26:35

Das Format kann getrennt für Datum und Uhrzeit mit der Funktion *SetNumberFormats* eingestellt werden.

Dasselbe gilt bei der Benutzung von Formatierungsfunktionen wie *DateStr*, *TimeStr* und *DateTimeStr*. Auch hier gelten die genannten Vorgaben und die Einstellung über *SetNumberFormats*. Allerdings kann man optional noch die Genauigkeit der Zeitangabe einstellen

von Stunden-genau bis Mikrosekunden-genau. Hier noch einige Beispiele:

Eingestelltes Format (Datum/Uhrzeit)	**Ausgabe**
Englisch/Englisch	12/23/2004 8:26:35.453 pm
International/International	2004-12-23 8:26
TurboDB/TurboDB	23.12.2004_08:26:35.453
Deutsch/TurboDB	23.12.2004 08:26:35.453
Deutsch/Deutsch	23.12.04 8:26:35,453
Englisch/International	12/23/2004 8:26:35.453
Englisch/International	12/23/2004 21:26:35.453

3.10.12 Berichte und Datenbankjobs starten

Wenn Sie von Ihrem TurboPL-Programm aus, einen Bericht oder Datenbankjob ausführen möchten, können Sie zwischen der Prozedur Run/*Drucken* und dem Prozedurpaar *OpenReport*/ *PrintDocument* wählen. Bei Run geben Sie einfach den Namen des gewünschten Ausgabedokuments an:

```
Run('TABELLE.Bericht1');
```

Dies entspricht einem Doppelklick auf das Element im Projektfenster, d.h. der Druckdialog wird angezeigt und der Anwender kann das Druckziel auswählen oder die Aktion abbrechen. Mit dem zusätzlichen Parameter *Modus* kann der Druckdialog auch unterdrückt werden, um das Dokument direkt auszudrucken oder in eine Datei zu schreiben. In diesem Fall gelten die letzten Einstellungen bzw. die mit *SetzeDrucker* und *SetzeAusgabeDatei* gemachten.

Mehr Kontrolle über die auszudruckenden Daten bei einem Bericht erlaubt das Prozedurpaar *OpenReport*/*PrintDocument*. Es erlaubt Ihnen, nach dem Öffnen des Berichts die Auswahl und die Sortierung der Datensätze einzustellen, bevor er gedruckt wird:

```
OpenReport('TABELLE.Bericht1');
SetSortOrder('Index1');
PrintDocument;
```

Diese Sequenz funktioniert ganz analog zum Öffnen eines modalen Formulars mit Voreinstellungen:

```
OpenForm('TABELLE.Formular1');
SetSortOrder('Index1');
ExecModal(...)
```

Im Unterschied zu *Run* ist die Sortierung hier dynamisch, d.h. durch Programmcode zur Laufzeit eingestellt. Achten Sie darauf, dass die obige Sequenz im Kontext der Applikation ausgeführt werden muss (d.h. in einem Applikations-Modul oder mit ausgeschalteter Option Formularkontext), weil sich der Aufruf von *SetSortOrder* sonst auf das Formular bezieht statt auf den Bericht.

3.10.13 Zugriff auf Dateien

TurboPL enthält Funktionen, mit denen Sie beliebige Dateien lesen und schreiben können. Dateien werden über Zahlen identifiziert, die beim Öffnen der Datei mit *Reset* (Öffnen zum Lesen) oder *Rewrite* (Öffnen zum Schreiben) zurückgegeben werden.

Dies ist ein Beispiel, wie eine neue Datei angelegt und beschrieben wird:

```
vardef DateiHandle: Integer;
DateiHandle := Rewrite('c:\temp\NeueDatei.txt');
Write(DateiHandle, 'Neue Datei');
Close(DateiHandle);
```

Das Argument von *Write* ist zwar immer ein String, aber weil Strings in TurboPL beliebige Codes enthalten dürfen, bedeutet das nicht, dass man nur Text-Dateien erzeugen und lesen kann. Das folgende Beispiel öffnet eine Datei und prüft, ob es eine Bitmap-Datei ist. Diese ist an den Werten B und M in den ersten beiden Bytes zu erkennen. Anschließend liest es die Größe des Bildes (in Bytes) aus der Datei und liefert diese Zahl als Ergebnis zurück:

```
procedure ReadBitmapSize(FileName: String): Integer;
   vardef Result: Integer;
```

```
    vardef FileHandle: Integer;
    vardef D: String;
    FileHandle := Reset(FileName);
    D := Read(FileHandle, 2);
    if D[1] = 'B' and D[2] = 'M'
       D := Read(FileHandle, 4);
       Result := (((Asc(D[4]) * 256) + Asc(D[3])) * 256 + Asc(D[2])) * 256 +
Asc(D[1]);
    else
       Result := 0;
    end;
    Close(FileHandle);
    return Result;
endproc;
```

Der Aufruf von *Close* ist wichtig, damit die Datei wieder freigegeben wird, ihre Angaben im Windows Explorer aktualisiert werden und andere Anwendungen wieder darauf zugreifen können. Zwar kümmert sich auch TurboPL selbst darum, dass alle geöffneten Dateien auch wieder geschlossen werden. Wenn Sie aber in Ihrem Programm vergessen, *Close* aufzurufen, ist der Zeitpunkt für das Schließen unbekannt.

Wenn Sie als Dateiname die Bezeichnung *RamText* verwenden, wird eine "Datei" geöffnet, die nicht auf der Festplatte sondern nur im Hauptspeicher existiert und dadurch sehr schnell ist. Über eine Speicherdatei kann man zum Beispiel viel Text zusammenfassen und dann mit *Text2Clip* in die Zwischenablage schreiben. Umgekehrt können Sie den Text mit *Clip2Text* aus der Zwischenablage in die Speicherdatei holen und von dort stückweise auslesen. Der Haupteinsatzzweck von Speicherdateien - die Behandlung von Zeichenketten mit mehr als 255 Zeichen - ist jedoch mit der Einführung von beliebig langen Strings weggefallen.l

Siehe auch

Dateien und Ordner in der TurboPL Referenz

3.10.14 DLL-Funktionen aufrufen

Durch die Deklaration mit *DllProc* können Sie von *TurboPL* aus eine Funktion in einer DLL (Dynamic Link Library) aufrufen. Dies ist in einigen Fällen nützlich:

- Wenn Sie eine Windows-Funktion benutzen möchten, für die es in *TurboPL* kein Äquivalent gibt. Ein Großteil der Windows-Funktionen sind in DLLs enthalten.
- Wenn Sie selbst eine komplexe Funktion in einer anderen Programmiersprache schreiben wollen (z.B. Delphi, C++) und diese dann von *TurboPL* aus aufrufen.
- Wenn Sie von einem Dritthersteller eine DLL erhalten, um z.B. Geräte wie Telefonanlagen u.ä. anzusprechen.

Damit *TurboPL* eine Funktion aus einer DLL aufrufen kann, muss es die folgenden Informationen haben:

1. Den Namen der DLL
2. Den Namen der Funktion
3. Die Datentypen für die Parameter und den Rückgabewert der Funktion.

Außerdem müssen die Funktionen in der DLL nach aktuellem Windows-Standard als *stdcall* deklariert sein.

Die benötigten Informationen erhält *TurboPL* durch eine spezielle Deklaration der DLL-Funktion als *TurboPL*-Prozedur mit *dllproc*. Ein Beispiel:

```
dllproc MyDllFunction(AString: String; var AInteger: Integer; AReal: Real):
Integer library 'MyDll.dll';
```

Einige Standard-Typen von TurboPL werden automatisch auf typische DLL-Funktions-Typen abgebildet. Wo das nicht möglich ist und wo der Automatismus nicht passt, kann man den Datentypen der DLL explizit angeben:

```
dllproc MyDllFunction(AString: String as LPStr; var AInteger: Integer as I8;
AReal: Real): Integer as I2 library 'MyDll.dll';
```

Dabei steht die Bezeichnung hinter dem as jeweils für den entsprechenden Datentyp in der DLL. In der folgenden Liste wird erklärt, welche DLL-Datentypen definiert werden können:

TurboPL Bezeichnung	Erklärung	Typen in C und C++	Typen in Delphi	Benötigter TurboPL Typ	Standard
LPStr	ANSI-codierte Zeichenkette	LPSTR, LPCSTR, char*	PChar, PAnsiChar	String	
LPWStr	Unicode-codierte Zeichenkette	LPWSTR, LPCWSTR, wchar_t*	PWideChar	String	X
I1	Ein-Byte Integer	signed char, char	ShortInt, Byte	Integer	
I2	Zwei-Byte Integer	short int, unsigned short int, WORD	SmallInt	Integer	
I4	Vier-Byte Integer	int, long, DWORD	LongInt, Integer	Integer	X
I8	Acht-Byte Integer	__int64	Int64	Integer	
R8	Acht-Byte Real	double	Double	Real	X

Die Datentypen in *TurboPL* und die Datentypen in der DLL müssen natürlich zusammenpassen. Zum Beispiel kann *TurboPL* keinen String-Parameter als I4-Parameter an die DLL-Funktion übergeben. In der obigen Tabellen sehen Sie deshalb auch, welchen *TurboPL*-Typ sie für jeden der möglichen DLL-Datentypen einsetzen müssen. Ein Kreuz in der Spalte Standard zeigt an, dass ohne die Klausel mit *as* dieser DLL-Typ angenommen wird, wenn in der *dllproc*-Deklaration der entsprechende TurboPL-Typ eingetragen ist.

Andere DLL-Datentypen als die in der Tabelle eingetragenen sind in DLL-Funktionen derzeit nicht möglich.

Wichtiger Hinweis:

Ab der Version 4 werden Strings Standard-mäßig als Unicode-Zeichenketten an die DLL übergeben, während sie bis einschließlich Version 3 als ANSI-Zeichenketten betrachtet wurden. Sie müssen also vorhandenen Code ändern, damit er weiter funktioniert. Dafür gibt es zwei Möglichkeiten:

1. Sie ergänzen ein *as LPStr* bei den String-Parametern der *dllproc*-Deklaration. Dadurch ist das alte Verhalten wiederhergestellt.
2. Sie verwenden einen Unicode-String in der DLL. Falls es sich bei der DLL-Funktion um eine Windows-Funktion handelt ist das einfach. Sie geben einfach die Variante mit dem W am Ende des Namens an statt die Variante mit dem A. Dies wird auch im folgenden Beispiel gezeigt.

Beispiel:

Wem die *TurboPL*-Funktion *Execute* nicht genug Optionen bietet, der kann auch die Windows-Funktion selbst aufrufen. Beachten Sie, dass die Version für Unicode (wide string) mit dem W am Ende des Namens verwendet wird:

```
dllproc ShellExecuteW(hWnd: Integer; lpOperation: String; lpFile: String;
lpParameters: String; lpDirectory: String; nShowCmd: Integer): Integer Library
'shell32.dll';

procedure ZeichenprogrammAufrufen

  ShellExecuteW(0, "", "c:\winnt\system32\mspaint.exe", "s:\emsdemo01.jpg",
"", 5);

endproc
```

Die folgende Variante funktioniert ebenfalls, führt aber dazu, dass alle Strings in ANSI übergeben werden und deshalb evtl. Schwierigkeiten mit Sonderzeichen aus fremden Sprachen entstehen:

```
dllproc ShellExecuteA(hWnd: Integer; lpOperation: String as LPStr; lpFile:
String as LPStr; lpParameters: String as LPStr; lpDirectory: String as LPStr;
nShowCmd: Integer): Integer Library 'shell32.dll';
```

Siehe auch:

DllProc

3.10.15 Ole-Automation verwenden

Ole-Automation ist ein Windows-Standard, der es ermöglicht mit anderen Anwendungen, sogenannten Automatisierungs-Servern zu kommunizieren. Programme, die Ole-Automation als Server unterstützen (zum Beispiel gehören alle Anwendungen von MS Office dazu), veröffentlichen Klassen mit Methoden, die man aus anderen Programmen aufrufen kann.

TurboDB Studio unterstützt das Schreiben von Automatisierungs-Clients durch den Typ *OleObject*. Ein Ole-Objekt wird mit dem veröffentlichten Namen des Automatisierungs-Servers erzeugt; anschließend kann man die entsprechenden Methoden aufrufen. Anwendungsmöglichkeiten sind zum Beispiel:

- Serienbrief mit Word drucken
- Excel-Rechenblatt mit Daten aus einer TurboDB Tabelle füllen und eine Grafik dafür erstellen
- Ein eigenes (z.B. mit Visual Basic oder Delphi) geschriebenes Programm fernsteuern

Hier ist ein Beispiel für die Kommunikation mit Excel:

```
procedure CreateWorksheet;
  vardef Excel: OleObject;
  Excel := CreateOleObject('Excel.Application');
  Excel.Visible := 'Wahr';
  Excel.Workbooks.Add
  Excel.Range(SYSTEM.Start-Zelle, SYSTEM.Ende-Zelle).Value := SYSTEM.Text;
endproc;
```

Die Excel-Anwendung wird hier sichtbar gemacht, um das Ergebnis direkt sehen zu können. Dies ist meistens nicht nötig und auch nicht erwünscht.Bedenken Sie auch, dass beim Übersetzen weder der Name der Automatisierungsklasse noch die Methoden und die Parameter geprüft werden können. Fehler bemerken Sie hier erst beim Ausprobieren. Wenn Sie die selbe Anwendung auf einem Rechner starten, auf dem kein Excel installiert ist, wird ein Laufzeitfehler ausgelöst.

Wie die Ole-Klassen von Excel bzw. des gewünschten Automatisierungs-Servers heißen und welche Schnittstellen sie anbieten, entnehmen Sie der Dokumentation des jeweiligen Produkts.

3.10.16 Makro-Fehler behandeln

Wenn bei der Ausführung einer Systemfunktion oder eines Systembefehls ein Fehler auftritt, wird dieser auf eine von zwei Arten gemeldet:

- Durch einen speziellen **Rückgabewert**. Dies ist immer dann der Fall, wenn der Fehler nicht besonders schwerwiegend ist und wenn er in einer Prozedur aufgetreten ist. Bei solchen Fehlern ist der Skript-Programmierer verantwortlich dafür, dass er den Fehlercode untersucht und entsprechend reagiert. Beispielsweise liefert die Funktion *FindFirstFile* den Wert -2, wenn das zu durchsuchende Verzeichnis gar nicht existiert.
- Durch einen **Systemfehler**. Das passiert dann, wenn der Fehler entweder so bedeutend ist, dass evtl. kein ordnungsgemäßes Weiterarbeiten möglich ist, oder wenn der Fehler an einer Stelle auftritt, wo kein Rückgabewerte verfügbar ist, z.B. bei einer Variablenzuweisung. Bei einem Systemfehler wird die Ausführung des Makros oder des Programms abgebrochen und der Fehler wird an der Benutzeroberfläche gemeldet. Ein Beispiel dafür ist, wenn *ReadRec* mit einer nicht existierenden Tabellenummer aufgerufen wird.

Auf Systemfehler reagieren

Wenn Sie nicht möchten, dass bei einem Systemfehler abgebrochen wird, können Sie den Abbruch unterdrücken und den Fehler durch eigenen Programm-Code behandeln. Dies geschieht dadurch, dass Sie einen Block mit *try* und *except* um die potentielle Fehlerstelle herum aufbauen.

```
try
   ..Hier könnte ein Fehler auftreten
except
   ..Hier wird weitergemacht, wenn der Fehler tatsächlich auftritt.
end;
```

Falls kein Systemfehler zwischen try und except auftritt, wird der Teil zwischen *except* und *end* übersprungen und hinter dem *end* mit der Ausführung fortgefahren. Sollte jedoch zwischen dem *try* und dem *except* ein Systemfehler eintreten, werden alle folgenden Befehle bis zum *except*

übersprungen und dann die Befehle zwischen *except* und *end* durchgeführt. Anschließend geht es in beiden Fällen hinter dem *end* weiter. Innerhalb des except-Blocks können Sie mit über das *Error* -Objekt die Nummer und die Beschreibung des Fehlers abfragen.

Ein Beispiel:

```
vardef N: Integer;
try
   N := Val(IrgendeinString);
   Message('Der Wert von ' + IrgendeinString + ' ist ' + Str(N));
except
   Message('Systemfehler ' + Str(Error.Nummer) + ' ist aufgetreten: ' +
Error.Message);
end;
N := 0;
```

Wenn jetzt der String *IrgendeinString* einen ungültigen Ausdruck enthält, dann wird die Ausführung in der Zeile mit *Val* abgebrochen und mit der Message im *except*-Block fortgesetzt. Wenn der String gültig ist, wird sein Wert in der oberen Message ausgegeben. In jedem Fall wird also genau eine Message ausgegeben und anschließend der Wert von N wieder auf 0 gesetzt.

Einer der Vorteile dieser Methode ist die Tatsache, dass man *try/except*-Blöcke verschachteln kann. Damit ist es möglich innerhalb eines inneren Abschnitts eine andere Fehlerbehandlung durchzuführen als im äußeren Abschnitt. Betrachten Sie diesen Code:

```
procedure InnereProzedur
..Anweisungsblock 1
try
   ..Anweisungsblock 2
except
   Message('Ein Fehler ist im Anweisungsblock 2 aufgetreten.');
end;
..Anweisungsblock 3
endproc;

procedure ÄußereProzedur
try
   InnereProzedur
except
   Message('Ein Fehler ist in Anweisungsblock 1 oder 3 aufgetreten.');
end;
endproc;
```

Hier gilt für den Anweisungsblock 3 automatisch die Fehlerbehandlung wie für den Anweisungsblock 1. Das bedeutet, dass Sie beim Schreiben der Prozedur *InnereProzedur* nicht darauf achten müssen, wie die Prozedur *ÄußereProzedur* die Fehler außerhalb von Anwendungsblock 2 behandelt.

Die alte Methode

In den Versionen vor TurboDB Studio wurde die Systemfehler-Behandlung mit *try/except* noch nicht unterstützt. Hier wurde mit dem Steuerbefehl EC gearbeitet. Je nach Einstellung des Steuerbefehls EC (error check) bricht die Programmausführung bei einem Systemfehler ab (EC 0, das ist die Vorgabe) oder es bricht nur die aktuelle Zeile ab und der Fehlercode steht anschließend in der System-Variablen Error/Fehler (EC 1). Die Bedeutung dieser Fehlercodes finden sie unter "Fehlermeldungen".

Achtung: Das Programm wird trotz EC 1 abgebrochen, wenn der Rückgabewert der fehlerhaften Prozedur nicht abgefragt wird.

Fazit: Die Verwendung dieser alten Methode der Fehlerbehandlung wird nicht mehr empfohlen.

Beispiele

Die Funktion *FindFirstFile* liefert unterschiedliche Fehlercodes, wenn sie nichts finden konnte. Löst aber keinen Systemfehler aus:

```
vardef Res: Integer;
Res := FindFirstFile("C:\MyDir\*.*", "DHS", FName, FSize, FDate, FTime,
FAttr, FFolder);
if Res >= 0
   ..Alles ok, weitermachen
else if Res = -1
   ..
```

```
else if Res = -2
  ..
else if Res = -3
  ..
else
  ..
end;
```

WriteRec dagegen, löst einen Systemfehler aus, wenn das Schreiben nicht möglich ist. Diesen Systemfehler kann man mit *try* und *except* abfangen.

```
try
  WriteRec(MYTABLE, 380);
except
  Message("Fehler " + Str(Error.Number) + " beim Schreiben des Datensatzes: "
+ Error.Message)
end;
```

Nach der alten Methode würde man folgendes schreiben:

```
vardef Res: Integer;
.EC 1
Res := WriteRec(MYTABLE, 380);
if Res <= 0
  Message("Fehler " + Str($Fehler) + " beim Schreiben des Datensatzes: ");
else
  .. Alles ok, weitermachen
end;
.EC 0
```

3.10.17 VDP-Anwendungen für TurboDB Studio anpassen

Wenn Sie vorhandene Programme von Visual Data Publisher für TurboDB Studio anpassen wollen, müssen Sie unter Umständen einige Änderungen vornehmen, die im folgenden beschrieben sind. Öffnen Sie ihr Projekt und stellen Sie sicher, dass beim Öffnen selbst keine Fehlermeldungen auftreten. Beantworten Sie die Frage, ob das Programm neu übersetzt werden soll mit ja. Falls nun Fehler beim Übersetzen auftreten, die unter Visual Data Publisher nicht aufgetreten sind, können Sie im folgenden Abschnitt nachschlagen, was die wahrscheinliche Ursache für diese Fehlermeldung ist und diese beheben.

Auch wenn Sie das Programm übersetzt haben und ausprobieren, können noch einige (sehr wenige) Fehlermeldungen auftreten, die Sie mit Visual Data Publisher nicht hatten. Diese Fehlermeldungen sind im übernächsten Abschnitt zusammen mit Hinweisen zur Behebung beschrieben.

Fehlermeldungen beim Übersetzen

Fehler 125

Einer Konstanten kann nichts zugewiesen werden.

In VDP 3 war folgende Sequenz möglich:

```
def k = a * b
k := 3;
```

Dabei wird k zuerst als Kurzfunktion definiert und gleich anschließend wie eine Variable mit einem Wert belegt. Diese Anweisungsfolge ist nicht sinnvoll und verursacht deshalb in TurboDB Studio einen Fehlermeldung. Grundsätzlich kann man einem Namen, der als Kurzfunktion definiert wurde nur mit einem weiteren *def* oder mit *var* einen neuen Wert zuweisen:

```
def k = a * b
var k = 3
```

würde also funktionieren, auch wenn auch hier die Definition der Kurzfunktion überflüssig ist. Sinnvoller wäre es, für die zweite Definition einen eigenen Namen zu benutzen.

Fehler 174

Der Bezeichner ist nicht eindeutig.

Dies bedeutet, dass Sie auf eine Tabellenspalte oder einen Variablen-Namen verweisen, der in der Anwendung öfter vorkommt.

```
vardef a: String;
a := Name;
```

löst beispielsweise diesen Fehler aus, wenn es zwei Tabellen mit einer Spalte namens Name gibt. VDP 3 hat hier eine dieser Tabellen ausgewählt (normalerweise die, die im Projekt als erstes kommt), aber das führt manchmal zu unerwarteten Ergebnissen (das Programm funktioniert nicht mehr, wenn man die Reihenfolge der Tabellen im Projekt ändert). Deshalb müssen Sie in TurboDB Studio in solchen Fällen, den Namen der Tabelle voranstellen, um die Fehlermeldung zu beseitigen.

```
vardef a: String;
a := KUNDEN.Name;
```

Fehlermeldungen zur Laufzeit

Fehler 118

Die externen Prozedur "<Prozedurname>" hat entweder eine Ausnahme ausgelöst oder die dllproc-Deklaration ist nicht korrekt.

Wenn Sie diese Meldung mit Visual Data Publisher nicht hatten, scheidet die erste Möglichkeit als Fehlerursache aus. In TurboDB Studio entspricht ein String in einer *dllproc*-Deklaration einem Unicode-String, in VDP 4 einem ANIS-String. Dadurch wird die Dll-Funktion nicht mehr korrekt aufgerufen. Die einfachste Variante, dies zu korrigieren, ist es die dllproc-Deklaration anzupassen.

```
dllproc MyProc(S: String): Integer library "MyDll.dll"
```

wird zu

```
dllproc MyProc(S: String as LPStr): Integer library "MyDll.dll"
```

Allerdings ist dies nicht die beste Lösung, weil dann kein Unicode-String verwendet wird, dieser aber der Standard unter Windows sein sollte. Besser ist es, die DLL so zu ändern, dass ein Unicode-String verwendet wird. Wenn die aufgerufene Dll eine Windows-Dll ist, geht das ganz einfach dadurch, dass die Version der Funktion mit dem Endbuchstaben W statt dem Endbuchstaben A gewählt wird.

```
dllproc GetTempPathW(L: Integer; var Buffer: String): Integer library
"kernel32.dll";
```

statt

```
dllproc GetTempPathA(L: Integer; var Buffer: String): Integer library
"kernel32.dll";
```

Siehe auch "DLL-Funktionen aufrufen"

3.11 Auf Datenbankebene programmieren

3.11.1 Datenbank- und Oberflächen-Befehle

Bei der Programmierung mit TurboPL muss man sich klarmachen, dass der Zustand der Datenbanktabellen auf Ebene der Datenbank selbst und der sichtbare Zustand in der TurboDB Studio - Oberfläche unterschiedlich sind. Wenn Sie zum Beispiel ein Fenster für die Tabelle A anzeigen und dann einen Bericht für die selbe Tabelle ausführen, werden in diesem Bericht die Datensätze der Tabelle durchlaufen. Natürlich sehen Sie davon an der Oberfläche nichts; der aktuelle Datensatz bleibt hier erhalten. Umgekehrt können Sie in einer Tabellenansicht durch die Datensätze laufen ohne dass in der Datenbank intern ein Satzzeiger verändert wird. Die Datensätze sind in der Oberfläche gepuffert.

Ähnliches gilt für die Markierungen. Sie können selbstverständlich mehrere Fenster für die selbe Tabelle öffnen und in jedem dieser Fenster unterschiedliche Markierungen setzen. Jedes Fenster hat also seine eigenen Markierungslisten und diese sind wiederum nicht die selben, die intern in der Datenbank verwendet werden.

Aus diesem Grund gibt es Funktionen, die sich entweder speziell an die Oberflächen-Elemente richten oder speziell an die internen Datenstrukturen. Sie heißen Datenbank- und Oberflächenfunktionen. Datenbankfunktionen können auch aufgerufen werden, wenn keine Oberfläche vorhanden ist, z.B. in einer Web-Server-Anwendung oder wenn Sie die TurboDB-Bibliotheken in einer anderen Entwicklungsumgebung als TurboDB Studio benutzen. Oberflächenfunktionen dagegen sind speziell auf TurboDB Studio zugeschnitten.

Hier ist eine Gegenüberstellung von wichtigen Daten- und Oberflächenfunktionen:

Datenbankfunktion	**Oberflächenfunktion**

[ReadRec](#) [ShowRec](#)
[SetMark](#) [PutStar](#)
[IsMark](#) [IsStar](#)
[NMarks](#) [StarNum](#)
[Access](#) [SetSortOrder](#)

Wann Sie nun Datenbank- und wann Oberflächenfunktionen einsetzen, ergibt sich aus dem bisher Gesagten:

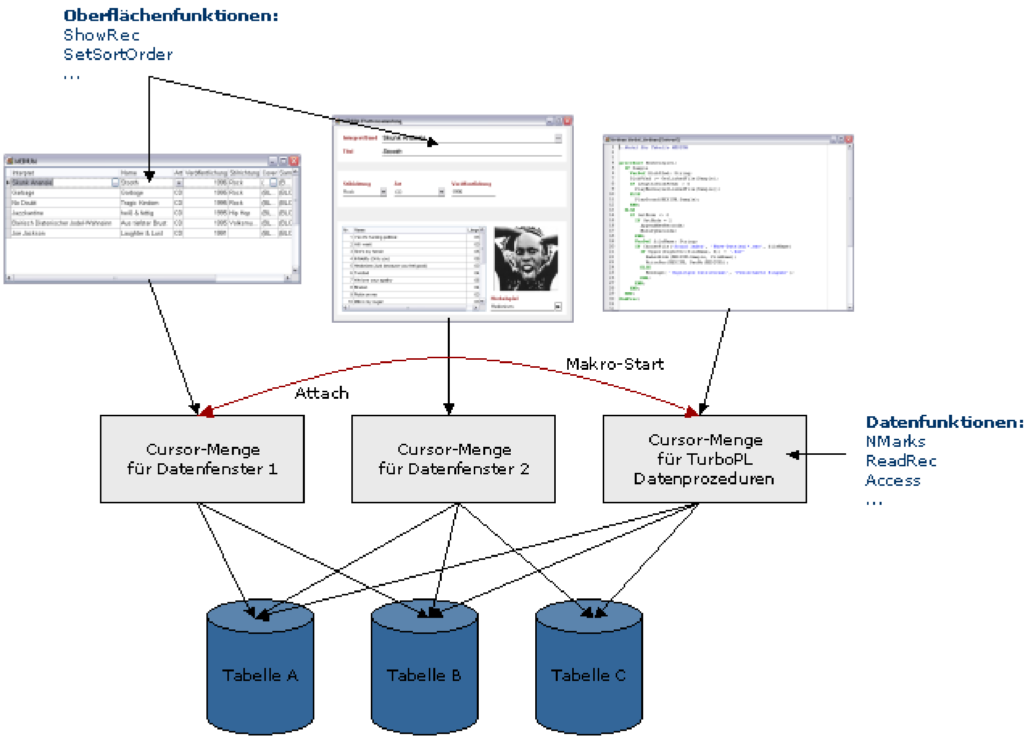
- Wenn Sie eine Funktionaliät anbieten wollen, die sich in der Benutzeroberfläche spiegelt, verwenden Sie Oberflächenfunktionen.
- Wenn Sie interne Auswertungen und Berechnungen implementieren, verwenden Sie Datenbank-Funktionen.

Es kommt natürlich auch vor, dass Sie einige umfangreiche Berechnungen durchführen, und das Ergebnis als Auswahl von Datensätzen in einer bestimmten Sortierung in einem Datenfenster präsentieren wollen. Auch in diesem Fall sollten Sie alle Datenbank-Operationen mit Datenbank-Funktionen durchführen. Um das Ergebnis dann in einem Datenfenster anzuzeigen, verwenden Sie die Datenfenster-Methode *Attach*. Diese Methode übernimmt die aktuellen Einstellungen für Tabellen, Markierungen und Satzzeiger aus der Datenbank in ein Datenfenster.

3.11.2 Das Programmiermodell von TurboPL

Das Programmiermodell

Bei der Programmierung mit TurboPL kann man sich den Aufbau einer Anwendung schematisch als drei Ebenen von Objekten vorstellen.



Die unterste Ebene sind die Tabellen. Sie enthalten die Daten inklusive der Indexe, Memos, Blobs und Relationsfelder. Tabellen werden als *.dat-Objekte gespeichert und bei Verzeichnisdatenbanken auch als dat-Dateien sichtbar.

In der mittleren Ebene greifen Cursor auf die Tabellen zu. Ein Cursor besitzt einen Satzzeiger, der

den aktuellen Datensatz der Tabelle spezifiziert, eine Markierungsliste (die sogenannte interne Markierungsliste) und den Zugriff. Der Zugriff wiederum legt fest, ob auf die Daten in der natürlichen Reihenfolge zugegriffen wird, über einen Index oder über die interne Markierungsliste. (Außerdem gibt es noch Filter und Bereiche für den Zugriff, aber die sind ein fortgeschrittenes Thema.) Für eine Tabelle kann es keinen, einen oder mehrere Cursor geben. Das ist wichtig, damit unterschiedliche Datenfenster für die selbe Tabelle unterschiedliche Satzzeiger, Zugriffe und Markierungslisten für diese Tabelle pflegen können.

Datenfenster sind also die dritte und oberste Ebene des Programmiermodells. Sobald ein Formular geöffnet wird, erzeugt es einen Cursor für die Primärtabelle, über den es auf die Daten der Tabelle zugreift. Wenn eingebettete Tabellenansichten vorhanden sind, erzeugen auch diese jeweils einen Cursor auf ihre zugehörige Tabelle. Jedes Datenfenster – egal ob das Formular selbst oder eine eingebettete Tabellenansicht – hat genau einen solchen Cursor.

Nur wenn die eingebettete Tabellenansicht die selbe Tabelle darstellt wie das Formular und sie außerdem entsprechend gekennzeichnet ist (Eigenschaft Tabelle ist Formular), dann verwendet diese eingebettete Tabellenansicht den selben Cursor wie das Formular. Deshalb hat sie immer den selben aktuellen Datensatz, die selbe Sortierung, usw.

Zu einem Formular gehört also immer mindestens ein Cursor, es kann aber mehrere davon besitzen. Aber nicht nur Formulare haben Cursor. Auch Berichte und Datenbankjobs haben welche, sonst könnten sie nicht auf die Daten der Tabelle zugreifen.

Zusätzlich zu all diesen Cursor-Mengen für Formulare (von denen keine vorhanden ist, wenn kein Formular geöffnet wurde), gibt es immer noch eine weitere für TurboPL-Datenbank-Befehle. Diese werden für Datenbankjobs verwendet, weil die ja auch ohne Formular ausgeführt werden können aber auch für Makros, damit diese nicht das Formular durcheinander bringen.

Zusammenhang zwischen Eigenschaften der Oberfläche und im Cursor

Auf den ersten Blick könnte man denken, was im Cursor eingestellt ist, sieht man auch an der Oberfläche. Das ist im Prinzip tatsächlich richtig: Das Datenfenster zeigt den aktuellen Datensatz des Cursors, die Reihenfolge der Datensätze entspricht dem im Cursor eingestellten Zugriff usw.

Bei einigen anderen Eigenschaften ist der Zusammenhang nicht so direkt. Zum Beispiel sieht man die Markierungen des Cursors nicht direkt sondern in der Auswahl der angezeigten Datensätze. Die sichtbaren Markierungen (die aus Tradition "Sternchen" heißen, auch wenn sie in TurboDB Studio als Kreise dargestellt werden) haben dagegen mit dem Cursor überhaupt nichts zu tun. Sie sind alleine eine Eigenschaft des Datenfensters.

Wenn der Anwender also mit der Maus sichtbare Markierungen setzt und anschließend den Befehl "Nur markierte Datensätze anzeigen" auswählt, passieren zwei Dinge:

1. Werden die sichtbaren Markierungen in interne verwandelt, d.h. das Datenfenster übergibt sie dem Cursor.
2. Der Zugriff im Cursor wird auf Markierung gestellt, was bedeutet, dass nur noch die Datensätze mit Markierung angezeigt werden.

Ein anderes Beispiel für einen komplexeren Zusammenhang zwischen Datenfenster und Cursor ist der Bearbeitungsmodus. Der Cursor verwaltet einen solchen Modus für den gerade aktuellen Datensatz, dieser wird zu einem bestimmten Zeitpunkt entweder betrachtet, geändert oder neu eingegeben. Die entsprechenden Schalter Datenfenster dagegen bedeuten, dass der Anwender ändern beziehungsweise neu eingeben darf. Beispielsweise kann man mit eingeschaltetem Neueingabemodus auch schon vorhandene Datensätze betrachten.

In diesem Fall ist der Modus im Datenfenster "Neueingabe erlaubt", der Bearbeitungs-Modus im Cursor jedoch "betrachten". Umgekehrt ist es in einer Prozedur möglich, den Datensatz mit einer Datenbank-Prozedur auf "ändern" zu schalten und per Programm zu editieren, während im Formular der Schalter Datensätze bearbeiten nicht aktiviert ist.

Datenbank-Befehle und Oberflächen-Befehle

TurboPL enthält sowohl Befehle, die auf die TurboPL-Cursor wirken als auch Befehle für die Datenfenster. Erstere heißen Datenbank-Befehle - ReadRec, SetMark, Access, Replace gehören zum Beispiel dazu - letztere Oberflächen-Befehle wie ShowRec, SetSortOrder und andere. Oberflächen-Befehle können natürlich nur dann ausgeführt werden, wenn mindestens ein Formular geöffnet ist. Den Effekt einer Oberflächenprozedur sieht man deshalb immer sofort im Datenfenster. Allerdings bewirken viele Oberflächen-Prozeduren auch eine Änderung am Cursor. Die Prozedur Sortierung zum Beispiel ändert als erstes den Zugriff des Cursors und zeigt dann anschließend die Daten neu an, damit die neue Sortierung sichtbar ist. Außerdem aktualisiert sie

die Anzeige der Sortierung im Kombinationsfeld in der Werkzeugleiste. Das bedeutet, dass nach der Ausführung eines Oberflächen-Befehls der Cursor und das Datenfenster entsprechend angepasst und synchronisiert sind.

Bei Datenbank-Befehlen ist das nicht so, weil sie auf den Cursor für TurboPL-Datenbefehle wirken und erst einmal nicht auf die Oberfläche. Das ist ein wichtiger Vorteil. Es würde zum Beispiel nicht gut aussehen, wenn eine Prozedur eine schnelle Schleife über viele Datensätze macht, um beispielsweise bestimmte davon zu markieren und das Datenfenster würde alle Datensätze auch tatsächlich darstellen. Abgesehen davon wäre das viel zu langsam.

Stattdessen gibt es einen expliziten Befehl, der die Cursor des Datenfensters wieder mit dem TurboPL-Cursor abgleicht, genauer gesagt, den Satzzeiger, die Markierungen und den Zugriff vom TurboPL-Cursor übernimmt. Der Befehl ist die Prozedur *Attach* und aktualisiert das Datenfenster inklusive etwaiger eingebetteter Tabellenansichten. *Attach* ruft man normalerweise als letzten Befehl in einer Prozedur auf, weil man ja vorher keine Aktualisierung im Datenfenster benötigt.

An dieser Stelle sollten wir noch einmal kurz auf den Unterschied zwischen der Prozedur Attach und der Prozedur Refresh eingehen. Refresh übernimmt weder Zugriff noch Markierung noch Satzzeiger aus dem Kursor, es zeigt einfach nur die Daten neu an um etwaige Änderungen in Datenfeldern an der Oberfläche wiederzuspiegeln. Refresh kann man jederzeit bedenkenlos aufrufen, weil es den Zustand des Datenfensters nicht ändert – ganz anders als Attach.

Was passiert aber nun, wenn man mit Datenbank-Befehlen den Zustand des Cursors ändert, ohne das Datenfenster mit Attach abzugleichen? Interessant ist das, wenn die Prozedur endet und das Datenfenster wieder die Kontrolle übernimmt oder wenn in der Prozedur eine modale Datenfenster-Funktion aufgerufen wird. Auch in diesem Fall ist das Datenfenster ja wieder aktiv und muss entscheiden welchen Zustand es anzeigt, seinen eigenen oder den des Cursors.

Im Prinzip ist die Antwort ganz einfach: Alle Änderungen auf dem Cursor werden verworfen und der ursprüngliche Zustand vor Aufruf der Prozedur wiederhergestellt. Da die Änderungen an der Tabelle aber verbleiben, gibt es hier ein paar Problemfälle.

- Der Datensatz, auf dem der Satzzeiger stand wurde inzwischen gelöscht: Dann steht der nach der Prozedur entweder auf dem darauffolgenden Datensatz oder auf dem letzten Datensatz der Tabelle.
- Ein Datensatz, der jetzt gelöscht ist, war markiert: Die Markierung wird entfernt

3.11.3 Datenbank-Programmierung und Oberflächenprogrammierung

TurboPL beinhaltet eine große Menge an Funktionen, mit denen Daten aus Tabellen bearbeitet werden können. Zu diesen Funktionen gehören unter anderen:

- *ReadRec*: Datensatz lesen
- *PostRec*: Datensatz schreiben
- *FindRec*: Datensatz suchen
- *SetMark*: Eine Markierung auf einen Datensatz setzen
- *NMarks*: Anzahl der Markierungen zurückliefern
- *ClearDat*: Eine Tabelle löschen
- *Access*: Sortierung festlegen
- und viele andere mehr. Diese Funktionen sind in der Referenz unter Datenbank-Befehle zusammengefasst.

Daneben gibt es eine andere Gruppe von Funktionen, die auf den ersten Blick eine ganz ähnliche Wirkung haben:

- *ShowRec*: Einen bestimmten Datensatz anzeigen
- *BySelection*: Datensätze suchen
- *PutStar*: Einen Datensatz kennzeichnen
- *SetSortOrder*: Sortierung festlegen

und eine ganze Reihe weiterer Befehle. Diese Funktionen sind in der Referenz unter Benutzerschnittstelle zu finden.

Zwischen diesen beiden Gruppen von Funktionen besteht aber ein fundamentaler Unterschied:

Datenbank-Befehle arbeiten rein auf den Daten-Tabellen also intern während Oberflächen-Funktionen rein auf den Formularen arbeiten und keinen Einfluss auf die Datenbank-Tabellen selbst haben.

Das kann man auch daran erkennen, dass Oberflächenfunktionen nur ausgeführt werden können, wenn ein passendes Formular geöffnet ist, und dass sie sich immer auf ein bestimmtes Formular beziehen. Datenbank-Befehle dagegen können immer ausgeführt werden, da sie in der Datenbank selbst ablaufen. Programme, die mit eine .NET-Sprache oder mit Delphi geschrieben sind und TurboDB einsetzen, können Datenbank-Befehle benutzten. Oberflächenfunktionen dagegen sind spezifisch für TurboDB Studio.

Den Unterschied zwischen den beiden Gruppen erkennt man auch an ihren Parametern: Datenbank-Befehle haben als ersten Parameter meistens einen Tabellennamen (manchmal einen Feldnamen). Oberflächenfunktionen haben keinen solchen Parameter. Sie beziehen sich entweder auf das gerade aktive Datenfenster in der Benutzeroberfläche oder werden als Methoden von Datenfenstern aufgerufen. Also:

```
ReadRec(MYTABLE, 3)
ClearDat(MYTABLE)
```

aber

```
ShowRec(80) ..Zeigt Datensatz 80 im aktiven Datenfenster an
TheDataWnd.ShowRec(80) ..Zeigt Datensatz 80 im Datenfenster TheDataWnd an
```

Jedes Formular legt für jede Tabelle, aus der es Daten anzeigt einen Cursor an. Das ist ein Puffer, in dem der aktuellen Datensatz gespeichert ist und auch die Sortierung und die Auswahl. Wenn für eine Tabelle drei Formulare geöffnet sind, dann gibt es dazu auch drei getrennte Cursors mit den jeweiligen Einstellungen. Das ist auch nötig, weil in diesen drei Formularen unterschiedliche Sortierungen, Auswahlen und Datensätze eingestellt werden können. Zusätzlich gibt es noch einen Cursor pro Tabelle für die Datenbank-Befehle. In der folgenden Grafik sind die Cursors blau dargestellt und die Datenbank-Tabellen grün. In der oberen Reihe sieht man zwei Formulare, die beide sowohl TAB1 als auch TAB2 darstellen sowie ein Programm mit Datenbank-Befehlen.

Die Folge dieser Zweiteilung ist, dass Sie Datenbank-Befehle und Oberflächenfunktionen nicht mischen können, auch wenn das manchmal naheliegend wäre. Die folgende Kombination ist zum Beispiel sinnvoll:

```
Access(MYTABLE, 'KundenNachName');
ReadRec(MYTABLE, FirstRec(MYTABLE));
```

Nach dieser Sequenz ist derjenige Datensatz aktuell, dessen Kundenname im Alphabet der erste ist. Dieser Datensatz ist deshalb aber noch nicht in einem Formular angezeigt. Sinnvoll ist auch das:

```
SetSortOrder('KundenNachName');
ShowRec(-1);
```

Hier wird der alphabetisch erste Datensatz im Formular zum aktuellen gemacht. Nicht sinnvoll ist dagegen:

```
Access(MYTABLE, 'KundenNachName');
ShowRec(-1);
```

Der erste Befehl arbeitet auf der Datenbank, der zweite auf dem aktiven Formular. Angezeigt wird der Datensatz, der bezüglich der für dieses Formular gültigen Sortierung als erster kommt. Diese Sortierung ist nur dann *KundenNachName*, wenn sie vorher vom Anwender oder durch das Programm mit SetSortOrder so eingestellt wurde. Ebenso wenig Sinn macht:

```
SetSortOrder('KundenNachName');
ReadRec(MYTABLE, FirstRec(MYTABLE));
```

Der gelesene Datensatz richtet sich wahrscheinlich nicht nach *KundenNachName*, sonder nach dem was zuvor mit Access eingestellt wurde.

Eine andere beliebte aber dennoch falsche Mischung ist:

```
BySelection('Betrag > 300000', 1, 1);
if NMarks(MYTABLE) > 0 ..NICHT DER GEWÜNSCHTE EFFEKT
  .. usw.
```

Weil die zweite Funktion auf einem anderen Cursor arbeitet als die erste, liefert *NMarks* nicht die Anzahl der in *BySelection* gefunden Datensätze. Manchmal stimmt die Zahl zwar trotzdem, weil intern Datenbank-Befehle zur Implementierung der Oberflächenbefehle benutzt werden, aber Sie können und dürfen sich nicht darauf verlassen.

Um die Programmierung von Anwendungen zu vereinfachen, gibt es zwei Verbindungen zwischen

den Formular-Cursors und der Datenbank:

1. Wenn Sie ein Makro aus einem Formular heraus aufrufen, werden alle Cursors dieses Formulars in die Cursors der Datenbank kopiert. Mit anderen Worten: Beim Start eines Makros, das aus einem Formular heraus aufgerufen wurde, finden Sie auf der Datenbank-Tabelle genau dieselben Einstellungen wieder, die auch im Formular vorhanden waren. Deshalb können Sie mit *NMarks* oder mit *GetAccess* abfragen, welche Einstellungen im Datenfenster vorhanden sind, obwohl sich diese Funktionen auf die Datenbank-Tabellen beziehen.
2. Die Oberflächenfunktion *Attach* übernimmt die Einstellungen der Datenbank-Cursors in das Formular, von wo aus es aufgerufen wurde. Das bedeutet, dass eine Kombination aus *Access* und *Attach* auf das Formular den selben Effekt hat wie ein *SetSortOrder*.

Nach dem bisher gesagten, gibt es drei verschiedene Muster für TurboPL-Prozeduren, die aus einem Formular aufgerufen werden:

1. Die Prozedur führt ausschließlich Oberflächenfunktionen aus: Dies ist der Fall, wenn die Prozedur durch Setzen von Sortierung und Auswahl eine Datensicht aufbaut, einen Bericht ausdruckt, Datensätze kontrolliert oder ähnliches tut.
2. Die Prozedur führt ausschließlich Datenbank-Befehle aus: Sie kann z.B. neue Datensätze eintragen, alte löschen oder vorhandene ändern oder eine Statistik über die Datenbank berechnen. In manchen Fällen resultieren diese Aktionen nicht in einem Zustand, der im Formular angezeigt werden muss.
3. Datenbank-Befehle gefolgt von einem Attach und evtl noch weiteren Oberflächenfunktionen: Dies ist der häufigste Fall, weil man auf diese Weise Änderungen an der Datenbank durchführen kann und das Ergebnis in einem Formular darstellen.

Wann verwendet man nun Datenbank-Befehle und wann Oberflächenfunktionen? Im Grunde ist die Antwort einfach und eine logische Folgerung aus dem bisher Beschriebenen. Wenn Sie einfache, Formular-bezogene Befehle ausführen wollen, sollten Sie Oberflächenfunktionen verwenden. Für alle umfangreicheren Änderungen an Datensätzen, komplexe Suchen, Aufbereiten von Berichten und Reports usw. sind Datenbank-Befehle die einzige sinnvolle Wahl. Um das Ergebnis einer solchen Aktion im Formular sichtbar zu machen, verwenden Sie *Attach*.

Wenn Sie diese Zusammenhänge etwas komplex finden, ist das nicht verwunderlich. Die Zusammenhänge sind komplex. Experimentieren Sie ein bisschen mit einem richtigen Projekt herum, dann wird es klarer werden.

3.11.4 Datensätze lesen

Wenn Sie in Ihrem eigenen Makro Datensätze aus einer Tabelle lesen wollen, handelt es sich oft darum, alle Datensätze der Tabelle durchzugehen. Dies können Sie z.B. folgendermaßen erreichen:

```
MoveBegin(MYTABLE);
while ReadNext(MYTABLE)
  ..Jetzt können Sie mit dem aktuellen Datensatz der Tabelle arbeiten, z.B.
ein Feld ausgeben
  Message(MYTABLE.Name);
end;
```

Eine Alternative dazu ist Folgendes:

```
.sub MYTABLE
  Message(MYTABLE.Name);
.endsub
```

In der ersten Variante haben Sie wesentlich mehr Kontrolle über den gesamten Vorgang, dafür ist die zweite Variante etwas eleganter.

Sollten Sie die Datensätze in einer vorgegebenen Sortierung durchlaufen wollen, können Sie folgendes tun:

```
SetAccess(MYTABLE, 'MYTABLE_Number_Index');
MoveBegin(MYTABLE);
while ReadNext(MYTABLE)
  ..Jetzt können Sie mit dem aktuellen Datensatz der Tabelle arbeiten, z.B.
ein Feld ausgeben
  Message(MYTABLE.Name);
end;
```

In dieser Variante muss es einen Index namens MYTABLE_Number_Index geben, der die gewünschte Sortierung beinhaltet. Das Kommando SUB kann hier noch etwas mehr:

```
.sub MYTABLE
  Message(MYTABLE.Name);
.endsub Name
```

Die Sortierung nach dem Datenfeld Name erfolgt hier unabhängig davon, ob ein passender Index existiert oder nicht. Allerdings ist die Ausführung natürlich schneller, wenn dies der Fall ist.

Analog zu *MoveBegin* und *ReadNext* gibt es auch die Befehle *MoveEnd* und *ReadPrev*, mit denen man die Datensätze der Tabelle von hinten nach vorne durchlaufen kann. Dies ist nützlich, wenn innerhalb des Durchlaufs Datensätze gelöscht werden, weil sich dies dann auf den weiteren Verlauf der Schleife nicht auswirken kann.

In älteren Versionen wurde *MoveBegin* und *ReadNext* noch nicht unterstützt. Das Lesen von Datensätzen sah damals folgendermaßen aus und funktioniert auch in TurboDB Studio noch:

```
vardef RecNo: Number;
SetAccess(MYTABLE, 'MYTABLE_Number_Index');
RecNo := FirstRec(MYTABLE);
while RecNo > 0
  ReadRec(MYTABLE, RecNo);
  ..Jetzt können Sie mit dem aktuellen Datensatz der Tabelle arbeiten, z.B.
ein Feld ausgeben
  Message(MYTABLE.Name);
  RecNo := NextRec(MYTABLE);
end;
```

Diese Methode ist erheblich umständlicher als die neue Methode mit *MoveBegin* und *ReadNext*. Außerdem wird hier in einer Mehrbenutzer-Version zusätzlich noch eine Sperre benötigt, damit die physikalische Satznummer garantiert gültig bleibt. Dies ist bei *MoveBegin* und *ReadNext* nicht nötig.

Siehe auch

MoveBegin, MoveEnd, ReadNext, ReadPrev, ReadRec

3.11.5 Datensätze zwischenspeichern

Mit Hilfe spezieller Datensatz-Variablen können Sie ganze Datensätze in Programm-Variablen zwischenspeichern:

```
vardef Rec: Record TABELLE;
```

definiert eine Variable, die genau einen Datensatz der Tabelle TABELLE aufnehmen kann. Mit den beiden Funktionen GetRec und PutRec kann dann der Satz von der Tabelle in die Variable bzw. von der Variablen zurück in die Tabelle geschrieben werden. Dieses Verfahren ist in vielen Fällen praktisch:

- Wenn Sie einen Datensatz sichern wollen, um eine spätere Version mit der Original-Version zu vergleichen.
- Wenn Sie einen Datensatz innerhalb einer Tabelle kopieren wollen.

Eng mit diesem Vorgehen verwandt ist die Funktion SetRecord, mit der ein Datensatz von einer Tabelle in eine andere kopiert werden kann. Damit können Sie Tabellen zusammenführen, abgleichen und kopieren sowie Daten aus einer Tabelle exportieren.

3.11.6 Umgang mit Memos und Blobs

Memos und Unicode Memos werden in TurboDB Studio grundsätzlich wie eine lange Zeichenkette behandelt. Das bedeutet, Sie können diese Felder wie String-Felder lesen und schreiben. Da Zeichenketten in TurboDB Studio praktisch beliebig lang sein können, passt der gesamte Inhalt eines Memo-Feldes in einen solchen String. Umgekehrt können Sie ein Memo dadurch schreiben, dass Sie einen String mit dem Inhalt an ein Memo-Feld zuweisen. Dasselbe gilt auch auf Unicode-Memos und Bilder/Klänge (Blobs). Nur die Prozedur GetField liefert aus Kompatibilitätsgründen nicht den Inhalt sondern wie bisher den Text (MEMO) oder (Memo) bzw. (BLOB) oder (Blob).

Darüberhinaus gibt es auch eine Reihe von speziellen Prozeduren für Memos und Blobs, die im Abschnitt "Memos und Blobs" beschrieben sind.

3.11.7 Datensätze löschen

Zum Löschen eines einzelnen Datensatzes benutzen Sie die Funktion *DelRec*:

```
while FileSize(MYTABLE) > 0
    DelRec(MYTABLE, 1)
```

Diese Routine löscht alle Datensätze der Tabelle MYTABLE einzeln. Das dauert ziemlich lange und ist nur zu Demonstrationszwecken gedacht. Sollten Sie tatsächlich die ganze Tabelle löschen wollen, verwenden Sie besser die Funktion *ClearDat*.

```
ClearDat(MYTABLE);
```

Beachten Sie, dass TurboDB den Platz gelöschter Datensätze sofort mit dem letzten Datensatz der Tabelle wieder auffüllt. Wenn Sie alle Datensätze einer Tabelle löschen wollen, in denen der Nachname *Maier* ist, dann ist der folgende Code nicht richtig:

```
vardef RecNo: Number;
RecNo := FirstRec(MYTABLE);
while RecNo > 0
    ReadRec(MYTABLE, RecNo);
    if MYTABLE.Nachname = 'Maier'
       DelRec(MYTABLE, RecNo);
    end;
    RecNo := NextRec(MYTABLE);
end;
```

Das funktioniert deswegen nicht, weil beim ersten *Maier*, der gefunden und gelöscht wird, der letzte Datensatz der Tabelle an seine Stelle verschoben wird. Dieser Datensatz wird aber nicht mehr untersucht. Falls dieser Datensatz also auch auf das Kriterium *Maier* passt, bleibt er verschont und die Tabelle enthält auch nach dem Durchlauf dieses Codes einen Datensatz mit dem Nachnamen *Maier*. Dieses Problem ist glücklicherweise einfach zu lösen:

```
vardef RecNo: Number;
RecNo := FirstRec(MYTABLE);
while RecNo > 0
    ReadRec(MYTABLE, RecNo);
    if MYTABLE.Nachname = 'Maier'
       DelRec(MYTABLE, RecNo);
       if RecNo > LastRec(MYTABLE)
          RecNo := 0
       end;
    else
       RecNo := NextRec(MYTABLE);
    end;
end;
```

Hier wird nur dann auf den nächsten Datensatz geschaltet, wenn der aktuelle Datensatz nicht gelöscht wurde. Dadurch wird im Falle des Löschens der an der Löschstelle eingesetzte Datensatz im folgenden Durchlauf der Schleife ebenfalls noch geprüft. Damit beim Löschen des letzten Datensatzes der Tabelle kein Fehler auftritt, wird *RecNo* nach dem Löschen auf 0 gesetzt, wenn das Ende der Tabelle erreicht ist.

Wesentlich einfacher und deshalb weniger fehleranfällig ist diese Variante:

```
MoveEnd(MYTABLE);
while ReadPrev(MYTABLE)
    if MYTABLE.Nachname = 'Maier'
       DelRec(MYTABLE);
    end;
end;
```

In dieser Variante löscht *DelRec* ohne Angabe der Satznummer den gerade aktuellen Datensatz der Tabelle. Das obige Problem mit den verschobenen Datensätzen, tritt hier gar nicht auf, weil die Tabelle rückwärts durchlaufen und keine physikalische Satznummer verwendet wird.

Wenn es darum geht, Datensätze zu löschen, die eine bestimmte Bedingung erfüllen, ist es effizienter, diese zuerst zu markieren und dann die markierten Datensätze zu löschen. Die beschriebenen Beispiele dienen nur zur Darstellung der Prinzipien. Sie werden auch dann interessant, wenn man keine Suchbedingung für die zu löschenden Datensätze angeben kann.

Siehe auch

Markierte Datensätze

3.11.8 Markierte Datensätze

Das Markieren von Datensätzen ist eine äußerst bequeme Methode, um mit Teilmengen Ihrer gesamten Daten zu arbeiten. TurboPL bietet effiziente und einfach zu verwendende Konstrukte, um Datensätze zu markieren und mit Listen aus markierten Datensätzen zu operieren. Diese Art mit Teilen der Gesamttabelle zu arbeiten, ist eine Alternative zu herkömmlichen Vorgehensweise mit SQL SELECT-Anweisungen, die in *TurboPL* auch möglich sind. Ein typischer Ablauf sieht so aus:

```
..Alle Personen markieren, die heute Geburtstag haben
LoopRecs(PERSON, Geburtstag = Today, SetMark(A, RecNo(A)));
..Die betrachtete Datenmenge auf diese einschränken
Access(A, "Markierung");
.. Alle markierten Datensätze durchlaufen
Rec := FirstRec(A);
while Rec > 0
    ..Hier mit dem Datensatz etwas ausführen
    Rec := NextRec(A);
end;
```

Mit den Prozeduren *GetMarks* und *PutMarks* können alle Markierungen einer Tabelle in ein Integer-Array gespeichert und wieder zurückgeschrieben werden. Beachten Sie dabei, dass in diesem Array dann physikalische Satznummern stehen, die sich durch Löschen von Datensätzen (zum Beispiel auch durch eine andere Anwendung) ändern können. Sie müssen also sicherstellen, dass zwischen dem Zeitpunkt von *GetMarks* und dem von *PutMarks* kein Datensatz aus der Tabelle gelöscht wird. Dies können Sie in einer Mehrbenutzer-Anwendung dadurch tun, dass Sie eine Schreibsperre auf die Tabelle setzen.

3.11.9 Datensätze schreiben

Datensätze das Ändern und Anfügen von Datensätzen geschieht jeweils in drei Stufen. Die erste Stufe startet die Aktion mit *NewRec* für neue Datensätze beziehungsweise *ModifyRec* zum Ändern eines vorhandenen Datensatzes. In der zweiten Stufe werden die einzelnen Felder belegt und in der dritten der Datensatz mit *PostRec* in die Tabelle geschrieben. Damit sieht das Anfügen eines Datensatzes folgendermaßen aus:

```
NewRec(MYTABLE);
MYTABLE.Vorname := 'Klaus';
MYTABLE.Nachname := 'Maier';
PostRec(MYTABLE);
```

Wenn das *PostRec* unterbleibt hat die gesamte Aktion keinen Effekt. Zum Ändern (im Beispiel des ersten Datensatzes) gehen Sie so vor:

```
MoveBegin(MYTABLE);
ModifyRec(MYTABLE);
MYTABLE.Vorname := 'Otto';
MYTABLE.Nachname := 'Kunze';
PostRec(MYTABLE);
```

Zu diesem Vorgehen gibt es eine Alternative, mit der Datensätze in einer einzigen Anweisung geändert oder angefügt werden können.

```
Replace MYTABLE(Vorname := 'Otto', Nachname := 'Kunze');
Append MYTABLE(Vorname := 'Klaus', Nachname := 'Maier');
```

In älteren Versionen waren die Funktionen *NewRec*, *ModifyRec* und *PostRec* noch nicht verfügbar. Hier gab es Datensatz-bezogene Funktionen, die auch in TurboDB Studio noch unterstützt werden. Bei diesen Funktionen wird als zweites Argument die physikalische Satznummer angegeben, deshalb ändert das folgende Beispiel den fünften Datensatz der Tabelle.

```
ReadRec(MYTABLE, 5);
MYTABLE.Vorname := 'Otto';
MYTABLE.Nachname := 'Kunze';
WriteRec(MYTABLE, 5);
```

Beachten Sie hier, dass die Satznummer bei *WriteRec* mit der Satznummer bei *ReadRec* übereinstimmt. Andernfalls werden unter Umständen wichtige Daten überschrieben.

Anders als bei *ModifyRec* sperrt *ReadRec* den gelesenen Datensatz nicht. Deshalb müssen Sie diese Variante in einem Mehrbenutzer-Umfeld noch mit einer expliziten Satzsperre versehen:

```
ReadRec(MYTABLE, 5);
EditOn(MYTABLE);
MYTABLE.Vorname := 'Otto';
MYTABLE.Nachname := 'Kunze';
WriteRec(MYTABLE, 5);
EditOff(MYTABLE)
```

Zum Anhängen eines neuen Datensatzes, lesen Sie in der ersten Version den Datensatz mit der virtuellen Nummer 0. Dadurch wird ein neuer Datensatz angelegt:

```
ReadRec(MYTABLE, 0);
MYTABLE.Vorname := 'Klaus';
MYTABLE.Nachname := 'Maier';
WriteRec(MYTABLE, LastRec(MYTABLE)+1);
```

Gegenüber der Methode mit *NewRec* ist der Nachteil hier, dass *ReadRec* mit der Datensatz-Nummer 0 weder Vorgabewerte noch eine Auto-Nummer in den Datensatz einträgt. Dadurch können Sie mit dem neuen Datensatz keine anderen Datensätze über Koppel- und Relationsfelder verknüpfen.

Siehe auch

NewRec, ModifyRec, PostRec, Replace, Append, ReadRec, WriteRec

3.11.10 Indexe verwalten

Indexe können auch in der Makrosprache erstellt und gelöscht werden.

Index erstellen

Hierfür steht die Funktion *GenIndex* zur Verfügung. Wenn Sie SQL bevorzugen, können Sie mit *ExecSQL* das Kommando *CREATE INDEX* ausführen.

Index löschen

Hierfür gibt es die Funktion *DelIndex* oder den Weg über SQL mit *DROP INDEX*.

Index wiederherstellen

Um einen Index zu reparieren, kann man entweder *RegenInd* oder *RegenAll* verwenden.

Vorhandene Indexe ermitteln

Mit den Funktionen *IndName* und *IndDef* kann man den Namen und die Definition eines Index bestimmen. Dabei werden die Indexe über eine Nummer identifiziert, die von -2 bis zur Anzahl benutzerdefinierter Indexe in der Tabelle geht. -2, -1 und 0 stehen dabei für den Zugriff Markierung, Zugriff Autonummer und den natürlichen Zugriff (ohne Index). Die Funktionen liefern einen Leerstring, wenn die höchste erlaubte Indexnummer überschritten wurde. Hier ist ein Beispiel, mit dem Name und Beschreibung aller Indexe einer Tabelle ausgegeben wird:

```
vardef Name: String;
vardef Def: String;
vardef i: Integer;
i := -2;
repeat
  Name := IndName(TABELLE, i);
  if Name <> ""
    Def := IndDef(TABELLE, i);
    Trace(Name + ": " + Def);
    i := i + 1;
  end;
until Name = ""
```

3.12 Anwendungen erstellen

Die Arbeit mit einem Datenbankprogramm ist oft darauf gerichtet, das erstellte Projekt für die tägliche Arbeit auch durch Dritte anzuwenden. In herkömmlichen Anwendungen müssen Sie dazu eine Applikation programmieren. TurboDB Studio bietet Ihnen mit dem User-Modus die Möglichkeit, ein von Ihnen erstelltes Projekt ohne jeden Programmieraufwand wie eine richtige Windows-Anwendung laufen zu lassen.

Die Applikation kann auf zwei Arten benutzt werden:

- Entweder starten Sie TurboDB Studio mit einer entsprechenden Kommandozeilen-Option. Dazu

benötigt jeder, der solche Applikationen einsetzen möchte eine Lizenz von *TurboDB Studio* (mindestens Standard). Der Entwickler einer solchen Lösung benötigt ebenfalls nur die Standard-Edition.

- Oder Sie erzeugen eine exe-Datei für die Applikation, die dann als eigenständiges Windows-Programm weitergegeben werden darf, wenn der Entwickler mindestens eine Professional-Edition besitzt. In diesem Fall benötigen die Anwender keine Lizenz von TurboDB Studio und Sie können ein vollwertiges Installationsprogramm für die Applikation generieren.

3.12.1 Den User-Modus aktivieren

Der User-Modus wird aktiviert, indem man beim Start des *TurboDB Studio* den Dateinamen eines Projektes mit übergibt und den Kommandozeilenparameter */P* oder */p* angibt. Die beiden Parameter stehen für den "großen" und den "kleinen" User-Modus.

Der kleine User-Modus entspricht dem Applikationstyp SDI (Einfensteranwendung), der große User-Modus dem Typ MDI (Mehrfensteranwendung).

So starten Sie TurboDB Studio im User-Modus aus dem Programm- oder Datei-Manager:

1. Wählen Sie *Datei/Ausführen* aus dem Menü.
2. Tragen Sie unter Befehlszeile den Pfad zu *TurboDB Studio* gefolgt vom Dateinamen der gewünschten Projektdatei und dem Parameter /P oder /p ein. Z.B.
   ```
   C:\Programme\dataWeb\TurboDB Studio\TurboDB Studio.exe C:\TdbsApps\Faktura.tdb
   /P
   ```

Entsprechend können Sie natürlich unter Windows eine Verknüpfung anlegen, so dass Ihre Anwendung im User-Modus gestartet wird.

3.12.2 Ein Projekt für den User-Modus vorbereiten

Im Grunde brauchen Sie für den User-Modus nichts vorzubereiten. Jedes mit *TurboDB Studio* erstellte Projekt kann im User-Modus ablaufen. Allerdings gibt es einige Möglichkeiten, Anwendungen für den "großen" User-Modus an Ihre Wünsche anzupassen.

So bestimmen Sie die Titel Ihrer Projektelemente für den User-Modus:

1. Der Name der Anwendung ist der Name des Projektes. Diesen können Sie im Projektfenster eingeben indem Sie den obersten Knoten im Projektbaum selektieren und die Eigenschaft Titel editieren.
2. Falls sich im Verzeichnis des Projektes eine Datei namens *Logo.bmp* befindet wird sie als Logo der Applikation verwendet.
3. Das jeweils erste Formular jeder Tabelle, das unter Verwendung *Daten bearbeiten* eingetragen hat (Eigenschaften des Formulars im Projektfenster) erscheint im User-Menü unter dem Menüpunkt *Tabelle*. Ein geeigneter Titel für ein solches Formulare ist also der vollständige Name der in der Tabelle enthaltenen Datensätze. Also z.B. *Buchungen* oder *Stichwörter*.
4. Alle Berichte und Datenbankjobs erscheinen im User-Menü unter dem Menü *Drucken*. Die Titel des Jobs oder Berichts sollten deshalb den Ausdruck selbst bezeichnen also z.B. *Rechnung*, *Saldenliste*, *Mahnungen* oder *Literaturverzeichnis*.
5. Unter *Ausführen* finden Sie im User-Menü alle Makros der Tabelle. Das sind diejenigen Prozeduren, die keine Parameter benötigen. Im Menü werden Unterstriche durch Leerzeichen ersetzt. Somit können Sie die Prozeduren z.B. *Bewegungsdaten_löschen*, *Nach_Stichwort_suchen* oder *Tagesabschluss_durchführen* nennen.

3.12.3 Eine Anwendung weitergeben

Turbo-Applikationen sind vor allem dann nützlich, wenn Sie von Dritten eingesetzt werden sollen. Der Anwender der Turbo-Applikation benötigt eine eigenen Lizenz des *TurboDB Studio*. Bedenken Sie bitte, dass Sie gegen Copyright-Gesetze verstoßen, wenn Sie Ihr Exemplar an Dritte weitergeben.

So installieren Sie eine Applikation für den User-Modus auf einem anderen Rechner:

1. Erstellen Sie ein Verzeichnis für die Applikation.

2. Kopieren Sie die Projektdatei sowie alle benötigten Tabellen und sonstigen Projektelemente in dieses Verzeichnis.
3. Wenn Sie Ihre Applikation mit einem eigenen Logo ausstatten wollen, kopieren Sie dieses als Bitmap-Datei unter dem Namen LOGO.BMP in das Verzeichnis.
4. Nun können Sie Ihre Applikation starten oder in den Programm- und/oder Dateimanager integrieren wie unter Den User-Modus aktivieren beschrieben.

Wenn Sie die professionelle Ausgabe von *TurboDB Studio* besitzen, können Sie auch eine eigene ausführbare Datei erstellen und diese weitergeben. Der Anwender benötigt dann keine Lizenz von *TurboDB Studio* oder von *TurboDB*.

So geben Sie die kompilierte Fassung Ihrer Anwendung weiter:

1. Erzeugen Sie die ausführbare Datei für die Anwendung *mit Datei/Anwendung erzeugen*.
2. Kopieren Sie die exe-Datei zusammen mit allen benötigten Projektelementen auf den Rechner des Anwenders.
3. Fügen Sie gegebenenfalls ein Start-Bild als Logo.bmp hinzu.
4. Legen Sie im Start-Menü des Anwendungsrechners einen Verweis auf die Anwendung an.

Die komfortabelste Möglichkeit ist schließlich ein eigenes Installationsprogramm für Ihre Anwendung; dieses wird im nächsten Abschnitt erläutert.

3.12.4 Ein Installationsprogramm für die Applikation erstellen

Wenn Sie ihre Applikation weitergeben oder verkaufen wollen, ist ein Installationsprogramm zwar nicht unbedingt nötig aber wünschenswert. TurboDB Studio unterstützt Sie bei der Erzeugung indem es eine Vorlage für das Freeware-Tool WiX (*Windows Installer XML*) erstellt, welches ohne weiteren Eingriff von Ihrer Seite ein vollständiges Standard Installationsprogramm für Windows (MSI) erzeugt.

WiX erfordert ein installiertes Microsoft .NET Framework auf Ihrem Computer. Sie können in der Windows *Systemsteuerung* unter *Software* überprüfen ob Sie das .NET Framework installiert haben. Wenn in der Liste der installierten Programme kein Eintrag für .NET Framework zu finden ist, laden Sie es von www.microsoft.com herunter und installieren sie es bevor Sie mit dem Erzeugen der Installationsvorlage beginnen. Die Anwender Ihrer Applikation benötigen .NET Framework nicht.

Die Unterstützung für Installationsprogramme ist in den Editionen ab Professional enthalten.

Um aus der Installationsvorlage ein Installationsprogramm zu erzeugen, müssen sie ein Programm installieren, das Dateien im WiX-Format verarbeiten kann. Es gibt eine ganze Reihe von kostenlosen Werkzeugen auf www.sourceforge.net, wir empfehlen derzeit WixEdit. Sie können dieses Tool in wenigen Minuten herunterladen und installieren.

So erstellen Sie ein Installationsprogramm für die Applikation mit WixEdit

1. Wählen Sie *Datei/Anwendung* erzeugen, um eine aktuelle Exe-Datei zu generieren.
2. Wählen Sie *Datei/Installationsvorlage* erstellen, um ein WiX-Projekt für Ihre Anwendung zu erstellen. Dabei wird im Anwendungsverzeichnis ein Unterverzeichnis namens Setup angelegt und darin eine Datei mit der Endung *wxs*.
3. Schließen Sie ihr Projekt in TurboDB Studio. WiX kann das Installationsprogramm nicht erzeugen, wenn die Dateien noch geöffnet sind: Das Hilfsprogramm *Light* verursacht in diesem Fall eine Fehlermeldung.
4. Starten Sie *WixEdit* aus dem Windows Startmenü.
5. Wählen Sie in WixEdit *File/Open* und suchen Sie die erstellte *wxs*-Datei für Ihre Applikation.
6. Jetzt können Sie Eigenschaften des Installationsprogramms abändern, wenn gewünscht. Allerdings ist das von TurboDB Studio erstellte Projekt vollständig und ohne weitere Änderungen lauffähig.
7. Wählen Sie *Tools/Wix Compile* im WixEdit-Hauptmenü, um das Installationsprogramm zu erzeugen und warten Sie auf die Abschlussmeldung *Finished in: X seconds*.
8. Im Unterverzeichnis Setup des Projektverzeichnisses Ihrer Applikation finden Sie jetzt eine msi-Datei zur Weitergabe.

3.12.5 Online-Hilfe erstellen

Ohne Online-Hilfe ist ein Programm keine echte Windows-Anwendung. Allerdings übersteigt der Aufwand, eine ausgefeilte Windows-Hilfe zu erstellen, den für das Umsetzen der eigentlichen Anwendung oft um ein Vielfaches. Es ist meist erheblich zeitaufwändiger, alle Funktionen einer Anwendung bis ins Detail zu beschreiben, als sie zu implementieren.

TurboDB Studio unterstützt zwei Arten der Online-Hilfe. Die klassische Windows-Hilfe (*.hlp) für Windows XP und das modernere HTML-Format (*.chm) für Windows VISTA.

Zum Erstellen der Hilfe kommen komfortable Werkzeuge zum Einsatz, auch kostenlose Freeware-Editoren wie etwa HelpMaker sind im Internet zu finden. Wir empfehlen KAL von Linn edv und Help & Manual von EC Software.

Die Arten der Hilfeunterstützung

TurboDB Studio stellt drei Möglichkeiten für die Hilfe zur Verfügung:

- keine Hilfe
- einfache Hilfe (nicht kontextsensitiv)
- kontextsensitive Hilfe

Die Art der Hilfe-Unterstützung legen Sie in den Projekt-Eigenschaften fest.

Für die Option *Keine* bleibt die Funktion der [F1]-Taste wirkungslos.

Standard Hilfe

Bei der *Standard*-Hilfe führt der Druck auf die [F1]-Taste immer zur ersten Seite der Hilfe-Datei (meist mit "Inhalt" bezeichnet).

Die Standard-Hilfe ist in den meisten Fällen ein sehr guter Kompromiss zwischen Anwendungskomfort und dem erforderlichen Entwicklungsaufwand. *TurboDB Studio* -Anwendungen sind oftmals so einfach und überschaubar, dass einige pauschale Hinweise zur Bedienung völlig ausreichen; eine detaillierte kontextsensitive Hilfe ist in vielen Fällen verzichtbar.

Bei Anwendungen mit Standard Hilfe wird bei Druck auf [F1] immer nur das erste Hilfe-Thema aufgerufen. Im einfachsten Fall steht hier nur ein klärender Text. Man kann aber auch eine ausführlichere Hilfedatei mit mehreren Themen erstellen, die dann über Hypertextlinks erreicht werden können. Schließlich bietet es sich noch an, die Hilfe-Themen stichwortartig aufzulisten. In diesem Fall wird die Suchen-Funktion der Hilfe aktiviert und man erhält eine recht komfortable und informative Hilfefunktion.

Um Ihren Anwendern diese Art der Online-Hilfe anzubieten ist es ausreichend eine Hilfedatei zu erstellen, diese genauso zu benennen wie Ihr TurboDB Studio Projekt und sie neben der erstellten Anwendung (Projekt.exe) zu speichern.

Kontextsensitive Hilfe

Die Option *Kontextsensitiv* legt fest, dass je nach gerade aktiver Programmfunktion ein spezieller Hilfetext aufgerufen wird. Wird der Kontext in der Hilfedatei nicht gefunden, erscheint eine entsprechende Fehlermeldung. In *TurboDB Studio* ist Kontext hier gleichzusetzen mit Formular. Das meint, es muss für jedes Formular eine Einsprungmarke in der Hilfe festgelegt werden. Die Namen der Einsprungmarken (ContextId) müssen mit ctxFrm beginnen. Darauf folgt der komplette Name des Formulares, also: *ctxFrmTABELLE.FORMULAR*.

Als Beispiel dient das Formular *Kraftfahrzeuge* der Tabelle *KFZ* aus dem Beispielprojekt *KFZ.* Die richtige ContextId sieht so aus: *ctxFrmKFZ.Kraftfahrzeuge*.

4 TurboPL Referenz

4.1 TurboPL Referenz

TurboDB Studio verfügt über eine mächtige Makrosprache namens TurboPL (vormals easy), die sowohl sehr effiziente Programmierung auf der Datenbank-Ebene und Tabelle-Ebene erlaubt, als auch eine komfortable Steuerung der Oberfläche.

In diesem Kapitel sind die Bestandteile der Makrosprache aufgelistet. Sie finden hier eine Referenz aller Funktionen, Kommandos und Steuerbefehle, die Sie benötigen, wenn Sie Module oder Datenbankjobs abfassen möchten. Ausführliche Erläuterungen zu diesen Themen enthält

das Benutzerhandbuch.
Im folgenden Abschnitt sind die TurboPL-Befehle nach Sachgebieten aufgelistet.

Grundlagen

Sprachkonzepte	Grundlengende Elemente der Sprache und ihre Syntax
Unit-Arten	Module, Klassen, Datenbankjobs

Basis-Funktionalität

Programmkontrolle	Prozeduren definieren, Schlüsselwörter für Schleifen und bedingte Verarbeitung
Steuerbefehle	Einstellungen für Compiler und Laufzeitumgebung
Zeichenketten	Zeichenketten vergleichen, Zeichenketten aufspalten, in Zeichenketten suchen
Mathematische Funktionen	Addieren, Runden usw.
Datum und Uhrzeit	Rechnen mit Datum und Uhrzeit
Array-Funktionen	Befehle zum Bearbeiten von Feldern
Dateien und Ordner	Dateien öffnen, beschreiben, lesen, schließen
Objekt-Funktionen	Programmieren mit Objekten
Statistik-Funktionen	Summen, Mittelwerte usw. berechnen
Diagnostische Funktionen	Fehlerbehandlung, Ausgaberoutinen zu Debugging-Zwecken

Datenbank

Tabellen	Tabellen anlegen, öffnen, Informationen über Tabellen
Datensätze manipulieren	Datensätze lesen, ändern, schreiben, anordnen...
Indexe verwalten	Indexe setzen, Informationen über Indexe
Interne Markierungen	Markieren, Markierungen sortieren, Markierungen speichern...
Netzwerk	Sperren einrichten, Anwenderverwaltung
Volltextsuche	Volltextindex erstellen, Volltextsuche durchführen

Datenbankjobs

Ausgabeformate	Formatierung der Ausgabe
Bereichs-Kommandos	Einteilung eines Datenbankjobs in Prolog, Kopf, Fuß, Daten und Gruppen
Befehle in Datenbankjobs	Kommandos und Steuerbefehle speziell für Datenbankjobs

Benutzerschnittstelle

Projekt-Verwaltung	Zugriff auf die Eigenschaften des Projektes und der Projektelemente
Mit Fenstern arbeiten	Meldungen ausgeben, Daten abfragen, Formulare öffnen und schließen...
Navigation im Datenfenster	Datensätze anzeigen, suchen
Oberflächenmarkierungen	Oberflächenmarkierungen bearbeiten
Druck-Funktionen	Ausdrucke starten, Druckerverwaltung

Datenaustausch und Multimedia

Dynamischer Datenaustausch	Funktionen zur DDE-Schnittstelle
Multimedia	Klänge und Videos spielen
Schnittstellen	Funktionen für die serielle Schnittstelle
Webserver-Funktionen	Funktionen für den Webserver-Betrieb
Zwischenablage	Schnittstelle zur Zwischenablage des Betriebsystems

4.2 Grundlagen

4.2.1 Ausdrücke

Bezeichner

- Als Bezeichner dürfen in Ausdrücken verwendet werden: Tabellenname, Feldnamen, für Auswahlfelder definierte Namen, vordefinierte Funktionen, vordefinierte Variablen (System-Variablen), eigene Prozedurnamen, eigene Variablennamen, Konstanten-Bezeichner
- Groß/Kleinschreibung ist relevant bei: Feldnamen, Variablennamen, selbstdefinierten Prozeduren.
- Keine Unterscheidung zwischen Groß- und Kleinschreibung wird gemacht bei: Kommandos, vordefinierten Funktionen, Tabellennamen
- Wenn ein Feld- oder Tabellenname mit einer vordefinierten Funktion kollidiert, drückt man mit einem vorangestellten Dollar-Zeichen $ aus, dass der Feld- oder Tabellenname gemeint ist.

Nummerische Ausdrücke

- Kommazahlen werden mit Dezimalpunkt geschrieben: 8.350
- Ein Datum in deutscher Schreibweise ist ein gültiger Ausdruck: 1.12.1999, ebenso eine Zeitangabe in 24-Stunden-Notation: 17:28.

Zeichenketten-Ausdrücke

- Für Zeichenketten-Literale können sowohl einfache als auch doppelte Anführungszeichen verwendet werden. Das öffnende und schließende Zeichen müssen aber gleich sein. Innerhalb der Zeichenkette kann dann das jeweils andere Anführungszeichen als Teil der Zeichenkette verwendet werden.

4.2.2 Variablen

Zur Variablendeklaration kennt TurboPL die folgenden Datentypen.

- String: Für beliebig lange Zeichenketten. Dieser werden intern als Unicode gespeichert.
- Integer: Ganze Zahlen im Bereich -9^63 bis +9^63 (d.h. -9223372036854775807 bis +9223372036854775807)
- Date: Tagesdatum
- Time: Uhrzeit
- DateTime: Kombination aus Datum und Uhrzeit
- Real (oder Number): Für Fließkommazahlen, die Rechengenauigkeit beträgt 15 Stellen inklusive Vorzeichen und Komma
- Object: Referenz auf ein Objekt (zum Beispiel Formular, eingebettete Tabelle, Bericht, Index).

Von diesen Typen können einzelne Variablen sowie Arrays (=Felder) angelegt werden. Zusätzlich gibt es den Typ RECORD(=Datensatz) zur Speicherung von Satzinhalten. Variablen werden mit vardef angelegt. In einem vardef können gleichzeitig mehrere Variablen des gleichen Typs definiert werden.

```
VarDef Einzelbetrag, Gesamtsumme: Real
VarDef Eingabe, Ausgabe, Zeile: String
```

Felder (Arrays)

Felder können bis zu 10 Dimensionen umfassen. In der Variablenanlage wird in eckigen Klammern der höchste Index der jeweiligen Dimension angegeben:

```
VarDef Vektor: Real[10]
VarDef Matrix: Integer[4,4]
```

Die einzelnen Feldelemente werden wiederum durch Indizierung in eckigen Klammern angesprochen. Der Index des ersten Feldelements ist immer 0. Beim Feld "Vektor" sind demnach folgende Feldelemente verfügbar:

```
Vektor[0], Vektor[1], ... , Vektor[10]
```

Mit der Standard-Funktion Redim kann die Dimension und der Rang des Arrays zur Laufzeit geändert werden, mit der Funktion High erhält man den Index des letzten Feldelements.

Ein Feld darf maximal 16 MB groß sein. Dies entspricht ca. 2 Mio. Einträgen bei Zahlen-Feldern und 4 Mio. Einträgen bei String-Feldern.

Bit-Felder (Bit-Arrays)

Eine besondere Rolle nehmen die Bit-Felder (Bit-Arrays) ein. Obwohl es den Datentyp Bit als solchen gar nicht gibt, kann man jedoch folgendes definieren:

```
VarDef Bits: Bit[100000]
```

Ein Eintrag in solches Array kann nur die Werte 0 oder 1 annehmen. Der Vorteil dabei ist, dass ein solches Array sehr platzsparend angelegt wird und dass logische Verknüpfungen mit solche Arrays durchgeführt werden können. Zusätzlich zu den Möglichkeiten mit normalen Arrays können Sie Bit-Arrays:

- in *GetMarks* und *PutMarks* verwenden, um die aktuellen Markierungen einer Tabelle zu lesen oder zu setzen,
- in den logischen Verknüpfungen *BitAnd*, *BitOr* und *BitAndNot* benutzen.

Obwohl es viele denkbare Einsatzbereiche für Bit-Arrays gibt, werden sie am häufigsten dazu benutzt, um verschiedene Markierungslisten schnell zu verknüpfen.

Initialisierung

Die Variablen, die innerhalb einer Prozedur angelegt werden, werden beim Aufruf der Prozedur folgendermaßen initialisiert:

String	""
Integer	0
Real	0.0
Date	Das fiktive Datum 0.1.0000
Time	00:00:00
DateTime	Der fiktive Zeitstempel 0.1.0000 0:00:00.000

Globale Variablen werden beim Laden des Programms ebenfalls auf diese Werte initialisiert. Änderungen bleiben aber bis zum Ende der Anwendung bzw. bis zum Neukompilieren oder bis zum Öffnen eines anderen Projekts erhalten. Um globale Variablen auf einen anderen Wert zu initialisieren, verwenden Sie die Prozedur *OnOpenProject*. Beachten Sie jedoch, dass diese im Entwickler-Modus nicht aufgerufen wird. Während der Entwicklung müssen Sie also ggf. diese Prozedur manuell aufrufen.

Eine weitere Möglichkeit, Einzelvariablen anzulegen, bietet das Kommando *LET*. Hier wird der Variablen direkt ein Wert zugewiesen. Aus diesem Grund ist eine Typangabe nicht erforderlich, da der Typ aus dem Wert bestimmt wird:

```
LET Ausgabedatei = "EXTERN.TXT"
LET PI = 3.1415
```

Verwendung

Der Zugriff auf den Wert einer Variablen erfolgt durch Angabe des Variablennamens. Wie aus der Syntaxbeschreibung ersichtlich, dürfen Variablen überall dort verwendet werden, wo auch Feldzugriffe oder Funktionen erlaubt sind.

Durch die Verwendung des Zuweisungsoperators ":=" können Variablen auch innerhalb von Ausdrücken verändert werden. In diesem Fall wird zunächst der auf den Zuweisungsoperator folgende Ausdruck berechnet. Dann wird dessen Wert in die Variable übertragen. Schließlich liefert der Variablenwert dann das Ergebnis der kompletten Zuweisung.

Im einfachsten Fall sieht eine Zuweisung dann folgendermaßen aus:

```
Bezeichner := Ausdruck
```

Es können jedoch auch mehrere Variablen gleichzeitig zugewiesen werden:

```
Startwert := Endwert := Durchschnitt := 100
```

Zuweisungen können auch irgendwo inmitten eines Ausdrucks stehen:

```
IF Text := Reset("C:\EXTERN.TXT") > 0 AND Zeile := ReadLn(Text) = "Code 123"
```

In diesem Beispiel wird im ersten Teil der Selektion die Variable *Text* belegt, im zweiten die Variable *Zeile*.

Satzpuffer

Zum Zwischenspeichern von Datensätzen einer Tabelle dienen die Record-Variablen. Mit dem Kommando

```
VarDef EinKunde: Record KUNDEN;
```

wird eine Variable angelegt, die genauso viele Bytes umfaßt wie ein Datensatz der Tabelle KUNDEN. Auf die einzelnen Felder des Datensatzes kann wie auf ein Real-Feld zugegriffen werden. Es ist nicht möglich ein Feld vom Typ Record zu verwenden. Das Lesen und Schreiben einer Datensatzvariablen erfolgt über die Funktionen GetRec und PutRec.

Konstanten

Zusätzlich zu Variablen gibt es noch die Möglichkeit Konstanten zu definieren. Diese können in Modulen wie Variablen eingesetzt werden, allerdings können keine neuen Werte zugeordnet werden. Zum Einsatz kommen Konstanten überall dort, wo der Umgang mit Variablen zweckmäßig ist, der Wert dieser Variablen sich im Programmablauf nicht ändert oder sich ändern darf.

```
Const Pi =  3.1415
Const CopyrightMsg = "Copyright 2000 by dataWeb GmbH"
```

Einen Wertetyp brauchen Sie dabei nicht angeben, dieser wird bei der Definition automatisch bestimmt. Sinnvollerweise sollten Konstanten am Beginn eines Modules definiert werden, da sie damit allen Prozeduren zur Verfügung stehen.

4.2.3 Objekte

Objektvariablen speichern Verweise auf komplexe Objekte, welche wiederum eine Anzahl von einfachen Variablen und auch Funktionen enthalten. Die im Programm verwendeten Objekte sind normalerweise nicht vom allgemeinsten Typ *Object* sondern von einem der spezielleren (abgeleiteten) Typen wie zum Beispiel *DataWnd* oder *Control*.

Die folgenden Objekttypen von TurboDB Studio können von TurboPL aus direkt angesprochen werden:

- Geöffnete Datenfenster. Dazu muss der Verweis auf das Datenfenster zuerst mit *FindDataWnd* gesucht und einer Objekt-Variablen vom Typ *DataWnd* zugewiesen werden. Dann können alle Oberflächen-Funktionen auf das Datenfenster angewendet werden.
- Eingebettete Tabellen. Auch hier wird zuerst mit *FindDataWnd* die eingebettete Tabelle gesucht und der Verweis auf die eingebettete Tabelle als *DataWnd* abgespeichert.
- Steuerelemente auf Formularen sind vom Typ *Control*.
- Berichte: Berichte sind unter ihrem vollständigen Namen direkt ansprechbar und können z.B. in *GetCompleteObjectName* verwendet werden: *GetCompleteObjectName(Project.KFZ.Begleitbrief)*.
- Indexe: Sind ebenfalls direkt ansprechbar. Ihr Name besteht aus dem Text Index_ und dem Namen des Index. Die automatisch angelegten Indexe ID und INR heißen Index_Standard und Index_RecordId. Sie können wie Berichte verwendet werden.

Den Typ *Object* selbst gibt es hauptsächlich deshalb, damit die anderen Objekttypen zusammengefasst werden. Eine direkte Anwendung in TurboPL-Programmen gibt es für diesen Typ praktisch nicht.

Anmerkung

Der Typ *OleObject* bezeichnet Ole-Automatisierungs-Objekte, die mit den eingebauten *TurboDB Studio*-Objekten wie zum Beispiel *DataWnd* nicht verwandt sind.

Siehe auch

Variable, DataWnd, Control, OleObject

4.2.4 Module

Ein Modul ist eine Datei, die Prozeduren enthält. In Modulen können alle Kommandos und vordefinierten Prozeduren aufgerufen werden. Es hat folgenden Aufbau:

```
..Zuerst kommt ein Kommentar, der den Zweck des Moduls erläutert

..Dann kommen evtl. nötige Steuerbefehle
.AK 0

..Falls Prozeduren aus anderen Modulen benutzt werden, hier mit uses einbinden
uses <AnderesModul1>
uses <AnderesMoul2>

..Wenn das Modul Konstanten benutzt, sollten die hier stehen
const Pi = 3.14159;
const MeinName = 'Hans-Dietrich';

..Jetzt kommen die Prozeduren
procedure ErsteProzedur;
   ..Hier wird was gemacht
endproc;


procedure ZweiteProzedur(a: Number; s: String);
   ..Hier wird was gemacht und dabei a und s verwendet
endproc;


procedure DritteProzedur(var a: Number; var s: String);
   ..Hier können a und s verändert werden
endproc;


procedure VierteProzedur(b: Number): Number;
   ..Diese Prozedur liefert ein nummerisches Ergebnis zurück
   return 18;
endproc;
```

Beachten Sie, dass eine Quellcode-Zeile in einem Modul bis zu 255 Zeichen umfassen darf. Längere Zeilen führen zu einer Fehlermeldung beim Übersetzen.

Eine formale Beschreibung des Moduls finden Sie in "Die Syntax von TurboPL".

4.2.5 Datenbankjobs

Ein Datenbankjob ist ein als Programm formulierter Datenbankbericht, in dem bei der Ausführung Datenbankfelder für Platzhalter eingesetzt werden. Darüberhinaus gibt es viele teilweise ausgeklügelte Anweisungen und Befehle, mit denen auch komplexe Auswertungen durchgeführt werden können.

Ein Datenbankjob ist entweder ein Serienbrief oder ein Report. Der Unterschied besteht nur darin, dass bei einem Serienbrief nach jedem Datensatz bzw. jeder Datensatzkombination eine neue Seite begonnen wird, beim Report dies unterbleibt. Die erste Zeile des Datenbankjobs legt fest, um was es sich handelt:

- Serienbriefe beginnen mit .LETTER oder ohne Angabe
- Reports beginnen mit .REPORT

Ein Datenbankjob hat folgenden Aufbau:

```
..Art des Datenbankjobs: Letter oder Report
.Report

..Steuerbefehle
.PL 66

.Prolog
..Ausgaben, die nur einmal zu Beginn des Datenbankjobs gemacht werden
```

```
.Kopf
..Ausgaben, die zu Beginn jeder Seite erscheinen

.Fuß
..Ausgabe, die am Ende jeder Seite erscheinen

.Gruppe
..Ausgaben, die für jede Gruppe einmal gemacht werden

.Daten
..Ausgaben, die für jeden Datensatz abgearbeitet werden

.Epilog
..Ausgaben, die nur einmal am Ende des Datenbankjobs gemacht werden
```

Die Reihenfolge der Bereiche muss eingehalten werden. Jeder Bereich kann aber auch weggelassen werden.

Beispiel für einen Serienbrief mit verschachteltem Unterreport:

```
.LETTER
.DATA
$KUNDEN.Vorname $KUNDEN.Name
$KUNDEN.Straße

$KUNDEN.PLZ $KUNDEN.Ort

Sehr geehrter Kunde,

leider müssen wir feststellen, dass folgende Rechnungen noch offen stehen:
.SUB NOT BESTELL.Bezahlt
      $(BESTELL.Datum:12 BESTELL.Betrag:12:2)
.ENDSUB BESTELL.Datum
Bitte begleichen Sie den Gesamtbetrag von DM $(SUM(BESTELL.Betrag):1:2) bis
zum $(DateStr(Today+14))...
```

Wie in Modulen dürfen auch die Zeilen in einem Datenbankjob 255 Zeichen nicht überschreiten.

4.2.6 Syntax-Beschreibung

Zur Beschreibung der Syntax der Makrosprache verwenden wir eine EBNF-Notation. Dabei gilt:

- Alles, was zwischen doppelten oder einfachen Anführungszeichen steht, muss exakt an der Stelle so (mit Ausnahme der Groß/Kleinschreibung) geschrieben werden.
- Sprachelemente in eckigen Klammern sind optional; sie können verwendet werden, müssen aber nicht.
- Sprachelemente in geschweiften Klammern können beliebig oft verwendet werden.
- Alternativen werden durch einen senkrechten Strich dargestellt. Dabei muss eine der Alternativen gewählt werden.
- Runde Klammern fassen Sprachelemente zusammen.
- Ist die Definition eines Sprachelements über mehrere Zeilen verteilt, so ist die Zeilenaufteilung Teil der Syntax. Das bedeutet, dass an diesen Stellen auch wirklich eine neue Zeile beginnen muss.

4.2.7 Die Syntax von TurboPL

```
Bezeichner::=Buchstabe.
Ganzzahl::=Ziffer{Ziffer}.
Zahlenkonstante::=Ganzzahl["."Ganzzahl].
Datumskonstante::=Ganzzahl"."Ganzzahl"."Ganzzahl.
Zeitkonstante::=Ganzzahl":"Ganzzahl.
Zahl::=Zahlenkonstante|Datumskonstante|Zeitkonstante.
Stringkonstante::=('"'{Zeichen}'"')|("'"{Zeichen}"'").
Konstante::=Zahl|Stringkonstante.
```

```
Dimensionsangabe::="[" Ganzzahl { "," Ganzzahl } "]".
Typ::=(("REAL" | "STRING")[Dimensionsangabe]) | "RECORD" Bezeichner.
Zeichen::=Buchstabe|Ziffer|"-".
Wort::=Zeichen{Zeichen}.
Ausdruck::=Term {("+"|"-") Term}.
Term::=Faktor {("*"|"/"|"DIV"|"MOD") Faktor}.
Faktor:=["+"|"-"](Konstante|Userdef|Funktion|("("Ausdruck")"))[Teilstring].
Teilstring::="["Ganzzahl [","Ganzahl "]".
Userdef::=Feldzugriff|Variable|Userfunktion.
Variable::=Bezeichner[":="Ausdruck].
Feldzugriff::=["$"]Bezeichner{"."(Bezeichner|Ganzzahl)}.
Variable::=Bezeichner.
Userfunktion::=Bezeichner["("Ausdruck{","Ausdruck}")"].
Selektion::=lTerm {"OR" lTerm}.
lTerm::=lFaktor {"AND" lFaktor}.
lFaktor::=["NOT"](lRelation|("("Selektion")").
lRelation::=Ausdruck{lOperator Ausdruck}.
lOperator::="AB"|"BIS"|"VON"|"IST"|"KLEINER"|"GRÖßER"|
      "ÄHNLICH"|"ENTHÄLT"|"HAT"|"IN"|"WIE"|
      "="|"<>"|"#"|"<="|">="|"?".
Indexbeschreibung::=Hierach-Index | Berechnet-Index.
Hierach-Index::=Single-Index{","Single-Index}.
Single-Index::=Bezeichner[":" Zahl].
Berechnet-Index::="("Ausdruck")"[":"Zahl]
Modul::={Prozedur}.
Prozedur::=
      "PROCEDURE" Bezeichner [Parameterliste] [":" ("STRING"|"REAL")]
           {Kommando}
      "ENDPROC".
Parameterliste::="(" Formalparameter {";" Formalparameter} ")".
Formalparameter::=["VAR"] Bezeichner {"," Bezeichner} ":" Typ.
Kommando::=
      Deklarationskommando | Ausführungskommando | Tabellenkommando.
Deklarationskommando::=Vardef|Let|Def|Const.
Vardef::="VARDEF" Bezeichner {"," Bezeichner} ":" Typ.
Let::="LET" Bezeichner "=" Ausdruck.
Def::="DEF" Bezeichner "=" Ausdruck.
Const::="Const" Bezeichner "=" Wert.
Ausführungskommando::=If | While | Repeat | Return | Report | Expressionlist.
If::=
      "IF" Selektion
           {Kommando}
      {"ELSIF" Selektion
           {Kommando}}
      ["ELSE"
           {Kommando}]
      "END".
While::=
      "WHILE" Selektion
           {Kommando}
      "END".
Repeat::=
      "REPEAT"
           {Kommando}
      "UNTIL" Selektion.
Return::="RETURN" [Ausdruck].
Report::="REPORT" Bezeichner.
Expressionlist::=Ausdruck { ";" Ausdruck }.
```

```
Tabellenkommando::=Relation | Filter | Sub | PrimtableIs | Setaccess| Replace
| Append.
Relation::="RELATION" [Reldef]{","Reldef}.
Reldef::=Statlink | Virttabelle | Zwangsverknüpfung.
Statlink::=Statlinkdef "=" Statlinkdef.
Statlinkdef::=Bezeichner"["Indexbescheibung"]"
Virttabelle::=Bezeichner"="Bezeichner.
Zwangsverknüpfung::=Bezeichner.
Filter::=    "FILTER" Indexexpression["/"Indexexprression].
Indexexpression::=Wort{","Wort}.
Sub::=
      "SUB" [Selektion]
           {Kommando}
      "ENDSUB" [Indexbeschreibung].
Primtable::="PRIMTABLEIS" Bezeichner.
Setaccess::="SETACCESS" Bezeichner.
Replace::="REPLACE" Satzbeschreibung.
Satzbeschreibung::=Bezeichner"("Feldzuweisung{","Feldzuweisung}")".
Feldzuweisung::=Bezeichner "=" Ausdruck.
Append::=    "APPEND" Satzbeschreibung.
```

4.2.8 Die Syntax von Datenbankjobs

In Datenbankjobs sind außer den normalen *TurboPL*-Elementen noch folgende zusätzliche Kommandos erlaubt:

```
Datenbankjob::=
      [Letter | Report]
      [Prolog]
      [Kopfbereich]
      [Fußbereich]
      [Gruppenbereich]
      [Datenbereich]
      [Epilog].
Letter::=[".LETTER"]
      Text.
Report::=".REPORT"
      Text.
Prolog::=
      ".PROLOGUE"
      Text.
Kopfbereich::=
      ".HEADER"
      Text.
Fußbereich::=
      ".FOOTER"
      Text.
Gruppenbereich::=
      ".GROUP" Ausdruck
      Text.
Datenbereich::=
      ".DATA"
      Text.
Epilog::=
      ".EPILOGUE"
      Text
Text::={Zeile}.
Zeile::=pKommando | Druckzeile.
pKommando::="."Kommando.
Druckzeile::={Zeichenfolge | Feldzugriff}.
Zeichenfolge::={Zeichen}.
```

```
Feldzugriff::="$"Ausgabekomponente.
Ausgabekomponente::=(Bezeichner{"."Bezeichner}) | ("("Ausgabeformat")").
```

4.3 Basis-Funktionalität

Die Kommandos und Prozeduren in diesem Abschnitt stellen die grundlegenden Mechanismen für die Programmierung mit TurboPL zur Verfügung.

4.3.1 Programmkontrolle

4.3.1.1 Programmkontrolle

In diesen Bereich fallen alle Kommandos und Funktionen, die den Ablauf eines Programms steuern, also zum Beispiel Schleifen und bedingte Ausführung.

as	Konvertiert einen Wert in einen anderen Datentyp.
Assigned	Prüft, ob eine Objekt-Variable belegt ist.
Choice	Wählt einen Wert aus einer Liste aus.
Compile	Übersetzt ein Modul oder einen Datenbankjob.
def	Definiert eine Kurzfunktion.
dllproc	Importiert eine Funktion aus einer Dynamischen Linkbibliothek.
end/ende	Beendet eine Schleife oder einen bedingten Block.
ExecProg	Führt eine als String gespeicherte Prozedur aus.
Execute	Führt ein anderes Programm aus.
for..next	Beginnt eine Programmschleife mit Zählvariable.
GetEnv	Liefert den Wert einer Umgebungsvariablen.
GetProductId	Liefert die Produkt-Id das ausführenden Programmes.
Halt	Beendet die Ausführung eines Makros oder eines Datenbankjobs.
if/falls	Beginnt einen bedingten Block.
include	Fügt beim Übersetzen eine andere Datei in die aktuellen Datei ein.
IsTask	Wird nicht mehr unterstützt.
Kommentare	Erlauben erklärenden Text in einem Modul oder Datenbankjob.
NLoop	Führt eine als Funktion geschriebene Programmschleife aus.
Note	Wird nicht mehr empfohlen.
ParamStr	Liefert eine Aufrufargument des Programms.
Sel	Wertet eine Bedingung aus und liefert das Ergebnis als Zahl.
Sleep/Pause	Wartet ein angegebene Anzahl von Millisekunden.
sub	Beginnt eine Schleife über die Datensätze einer Tabelle.
repeat/wiederhole	Beginnt eine Schleife, deren Abbruchbedingung am Ende steht.
return	Verlässt eine Prozedur.
uses	Verweist beim Übersetzen auf ein anderes Modul.
while/solange	Beginnt eine Schleife mit der Abbruchbedingung am Anfang.

4.3.1.2 as

Syntax

```
<Wert> as <Variablentyp>
```

Erklärung

Damit kann ein Variablenwert in einen Wert eines anderen Typs umgewandelt werden. Dieses Schlüsselwort ist nur im Modus SV 1 sinnvoll, weil im Modus SV 0 Variablen grundsätzlich

automatisch umgewandelt werden. Die Umwandlung der anderen Typen in einen String liefert grundsätzlich die Standard-Darstellung, wie sie auch in TurboPL-Programmen verwendet wird. Umgekehrt setzt die Umwandlung eines Strings in einen anderen Typ diese Standard-Darstellung voraus. Die übrigen Kombinationen sind im Folgenden aufgelistet:

Von	Nach	Ergebnis
Real	Integer	Gerundeter Wert
Real	Date	Datum, das der Tageszahl entspricht
Integer	Date	Datum, das der Tageszahl entspricht
Real	DateTime	Zeitstempel inkl. Uhrzeit, der der Tageszahl entspricht
Integer	DateTime	Zeitstempel, der der Tageszahl entspricht, die Uhrzeit ist 0:00
Real	Time	Uhrzeit, die der Minutenzahl seit Mitternacht entspricht inkl. Sekundenanteil
Integer	Time	Uhrzeit, die der Minutenanzahl seit Mitternacht entspricht, Sekunden sind 0.000
Time	Integer	Minutenanzahl nach Mitternacht gerundet
Time	Real	Minutenanzahl nach Mitternacht inkl. Bruchteile von Minuten
Time	Date	Nicht möglich
Time	DateTime	Zeitstempel am 0-ten Tag der Zeitrechnung
Date	Integer	Anzahl Tage seit dem Beginn der Zeitrechnung
Date	Real	Anzahl Tage seit dem Beginn der Zeitrechnung
Date	Time	Nicht möglich
Date	DateTime	Zeitstempel mit Uhrzeit 0:00
DateTime	Integer	Anzahl Tage gerundet
DateTime	Real	Anzahl Tage mit Bruchteil für Uhrzeit
DateTime	Time	Uhrzeit-Anteil des Zeitstempels
DateTime	Date	Datums-Anteil des Zeitstempels

Beispiel

Wenn Sie das folgende Beispiel beim Debuggen in der Anzeige der lokalen Variablen betrachten, sehen Sie für a den nummerischen Wert, während für d das entsprechende Datum angezeigt wird.

```
.SV 1
vardef d: Date;
vardef a: Integer;
a := 732308;
d := a as Date;
```

4.3.1.3 Assigned Prozedur

Syntax

```
Assigned(Object: Object): Integer
```

Erklärung

Liefert 1, wenn die Objektvariable belegt ist und ansonsten 0.

Beispiel

Der Entwickler will überprüfen, ob ein bestimmtes Formular geöffnet ist und dieses dann schließen

```
procedure FormularSchliessen(Formularname: String)
    vardef o: DataWnd;
    o := FindDataWnd(Formularname)
    if Assigned(o)
       Message("Das Formular " + Formularname + " wird geschlossen",
"Schließen")
       o.CloseWnd
    end
endproc
```

Siehe auch

FindDataWnd

4.3.1.4 Choice Prozedur

Syntax

```
Choice(Selector: Integer; Expression1, Expression2, ..., ExpressionN): Variant
XWert(Selektor: Integer; Ausdruck1, Ausdruck2, ..., AusdruckN): Variant
```

Erklärung

Die Funktion liefert den Ausdruck an der von Selektor bestimmten Stelle zurück. Wenn Zahl nicht im Bereich von 1..N liegt, liefert die Funktion als Ergebnis immer den letzten Ausdruck AusdruckN. Je nach Typ des Ausdrucks liefert die Funktion entweder einen String oder einen Zahltyp zurück.

Erst wird der bummerische Ausdruck Selector berechnet (ergibt n). Im Anschluß daran wird der n-te Ausdruck berechnet und als Ergebnis dieser Funktion zurückgeliefert. Falls kein n-ter Ausdruck vorhanden ist, wird der letzte Ausdruck der Reihe berechnet. Es muss mindestens ein Ausdruck in der Liste vorhanden sein, sonst wird ein Übersetzungsfehler ausgelöst.

Achtung

In VDP 3 wurden xWert-Ausdrücke ohne Ausdruck in der Liste übersetzt. Das Ergebnis war aber undefiniert.

Beispiel

```
Choice(x, "eins", "zwei", "drei")     ->  "eins" für x = 1
                                ->  "zwei" für x = 2
                                ->  "drei" in allen anderen Fällen
Choice(5, "eins", "zwei", "drei")     ->  "drei"
```

Sehr praktisch ist diese Funktion beispielsweise bei der Auswertung von Auswahlfeldem. Intern haben Auswahlfelder den Wert 0, wenn sie ganz leer sind, ansonsten eine 1, wenn der erste Wert gewählt ist, eine 2 beim zweiten Wert usw. Im folgenden Beispiel ist "Geschlecht" ein Auswahlfeld mit den Konstanten "weiblich", "männlich" und "unbekannt".

```
Choice(Geschlecht, "Sehr geehrte Frau " + $Name, "Sehr geehrter Herr " +
$Name, "Sehr geehrte Damen und Herren")
```

Siehe auch

if/else/end

4.3.1.5 Compile Prozedur

Syntax

```
Compile(Modul: String): Integer;
```

Erklärung

Mit *Compile* wird zur Laufzeit ein Modul in eine PRG-Datei übersetzt.

Erfolgte die Übersetzung fehlerfrei wird eine 0 zurückgegeben, sonst eine Fehlernummer. Besonders Hilfreich ist diese Funktion für große Projekte mit entsprechend vielen Modulen. Um das Projekt schnell auf den neusten Stand zu bringen, schreiben Sie eine Prozedur in der alle Module in der richtigen Reihenfolge übersetzt werden.

Beispiel

```
procedure CompileAll;
    ShowWait( 'Compiliere KUNDEN.MOD...');
    if Compile( 'KUNDEN.MOD') <> 0
       if Message( 'Fehler in Modul: KUNDEN', 'Compile', 2) = 2
          HideWait;
          Halt;
       end;
    end;
    ShowWait( 'Compiliere KFZ.MOD...');
    if Compile( 'KFZ.MOD') <> 0
       if Message( 'Fehler in Modul: KFZ', 'Compile', 2) = 2
          HideWait;
          Halt;
       end;
```

```
    end;
    HideWait
    Message('Projekt erfolgreich übersetzt');
endproc
```

4.3.1.6 def Kommando

Syntax

```
def Funktionsname = Ausdruck
```

Erklärung

Das Kommando *def* erlaubt die Definition von parameterlosen Kurzfunktionen. Solche können vorteilhaft zu Abkürzungen eingesetzt werden, wenn beispielsweise ein Wert aus mehreren Datenfeldern berechnet wird.

Die Syntax gleicht der des Kommandos *let*, nur dass eben keine Variable mit dem berechneten Wert angelegt wird, sondern hier wird vielmehr die Rechenvorschrift selbst gespeichert. Aus diesem Grund können Kurzfunktionen wie Datenfelder behandelt werden und dürfen beispielsweise als Sortierkriterien in Subreports verwendet werden.

Beispiel

```
def I0 = Italic(0)
def I1 = Italic(1)
def Differenz = SALDO95.Gesamtsumme - SALDO94.Gesamtsumme
sub Differenz >= 0  .. Nur die Satzkombinationen mit positiver Differenz
    ...
endSub
let Gesamtdifferenz = Sum(Differenz)
```

4.3.1.7 dllproc Kommando

Syntax

```
dllproc Funktionsname([Funktionsparameter: Parametertyp]): [Ergebnistyp]
library Dll-Dateiname
```

Erklärung

Definiert den Zugriff auf eine in einer Dll-Datei vorhandenen Funktion die Sie in einer anderen Programmiersprache wie z.B. C++ oder Delphi erstellt haben. Diese können danach wie die vorhandenen *TurboPL*-Funktionen in Modulen benutzt werden.

Wenn Sie eigene Dll-Funktionen für *TurboDB Studio* entwickeln wollen, müssen Sie folgende Punkte beachten:

- Der Aufruf der Funktion muss als *stdcall* definiert sein.
- Exportieren Sie Ihre Funktion unter dem Namen, den Sie unter *TurboDB Studio* verwenden wollen und achten Sie dabei auf die Groß/Kleinschreibung.
- Real-Zahlen werden als Double übergeben, Integer-Zahlen als 32-Bit Integer und Zeichenketten als Unicode-Strings.

Achtung

Ab TurboDB Studio müssen Zeichenketten als Unicode-Strings übergeben werden, während frühere Versionen Ansi-Strings verwendet haben. Deswegen müssen Sie bei Aufrufen in Windows-Bibliotheken die Unicode-Versionen dieser Funktionen verwenden, die sich durch ein nachgestelltes W auszeichnen. Z.B. ShellExecuteW (und nicht mehr wie bei VDP 3 ShellExecuteA). Selbstgeschriebene Funktionen müssen für String-Parameter die Werte als PWideChar (Delphi), wchar_t* (C++), TCHAR*/LPCTSTR* (C++ mit Compiler-Option UNICODE) erwarten.

Beispiel

Dieser Delphi - Quelltext zeigt die prinzipielle Erstellung einer Dll und deren Verwendung mit *TurboDB Studio*. Die Funktionen der Dll sind bewusst einfach gehalten, um die Möglichkeiten der Parameterübergabe zu verdeutlichen.

```
// Delphi - Quelltext
library TurboDBStudioDemoDll;

uses
  SysUtils,
```

```
  StdCtrls,
  Dialogs;

// Eine Meldung am Bildschirm ausgeben
procedure HelloWorld; stdcall; export;
begin
   ShowMessage('Hello World!');
end;

// Inhalt eines Strings oder Memofeldes in eine Datei schreiben
procedure WriteMemoToFile(Memo: PChar); stdcall; export;
var m: TMemo;
begin
   m := TMemo.Create(nil);
   m.Text := Memo;
   m.Lines.SaveToFile('TurboDB Studio Memo.txt');
   m.Free;
end;

// einen Integerwert zurückgeben
function GetIntResult: Integer; stdcall; export;
begin
   Result := 12345;
end;

// Übergebener Integer-Parameter wird verdoppelt
function GetIntParameter(MyParameter: Integer): Integer; stdcall; export;
begin
   Result := MyParameter * 2;
end;

// Übergebener Real-Parameter wird verdoppelt
function GetRealParameter(MyParameter: Double): Double; stdcall; export;
begin
   Result := MyParameter * 2;
end;

// Länge eines Strings bestimmen
function GetStringLen(const MyParameter: PWideChar): Integer; stdcall; export;
begin
   Result :=StrLen(MyParameter);
end;

// Speichergröße eines Strings bestimmen
function GetStringSize(const MyParameter: PChar): Integer; stdcall; export;
begin
   Result := sizeof(MyParameter);
end;

// Testen ob ein String leer ist
function IsStringEmpty(const MyParameter: PChar): Integer; stdcall; export;
begin
   if StrLen(MyParameter) = 0 then
      Result := 1
```

```
   else
      Result := 0;
end;

// Einen String mit Text belegen
procedure SetTurboPLString(MyParameter: PChar; MaxLen: Integer); stdcall;
export;
begin
   StrPLCopy(MyParameter, 'Der dataweb TestString', MaxLen);
end;

// Verdoppelt übergebenen Integer-Wert
procedure SetIntParameter(var MyParameter: Integer); stdcall; export;
begin
   MyParameter := MyParameter * 2;
end;

// Verdoppelt übergebenen Real-Wert
procedure SetRealParameter(var MyParameter: Double);stdcall;export;
begin
   MyParameter  := MyParameter * 2;
end;

// Vergleicht 2 Strings unabhängig von Groß/Kleinschreibung
function StrEqual(const param1, param2: PChar): Integer; stdcall; export;
begin
   Result := StrIComp(param1, param2);
end;

exports
   HelloWorld index 1,
   GetIntResult index 2,
   GetIntParameter index 3,
   GetRealParameter index 4,
   SetIntParameter index 5,
   SetRealParameter index 6,
   IsStringEmpty index 7,
   GetStringLen index 8,
   GetStringSize index 9,
   StrEqual index 10,
   SetTurboPLString index 11,
   WriteMemoToFile index 12;
begin
end.


..Dieses Modul greift auf die Dll - Funktionen zu
.SV 1

dllproc HelloWorld  library "TurboDBStudioDemoDll.dll";
dllproc GetIntResult: Integer  library "TurboDBStudioDemoDll.dll";

dllproc GetIntParameter(myParam: Integer):Integer  library
"TurboDBStudioDemoDll.dll";
dllproc GetRealParameter(myParam: Real): Real  library
```

```
"TurboDBStudioDemoDll.dll";

dllproc SetIntParameter(var myParam: Integer)  library
"TurboDBStudioDemoDll.dll";
dllproc SetRealParameter(var myParam: Real)  library
"TurboDBStudioDemoDll.dll";

dllproc SetTurboPLString(var myParam: String as LPStr; maxlen: Integer)
library "TurboDBStudioDemoDll.dll";
dllproc IsStringEmpty(myParam: String as LPStr): Integer  library
"TurboDBStudioDemoDll.dll";
dllproc GetStringLen(myParam: String as LPStr): Integer  library
"TurboDBStudioDemoDll.dll";
dllproc GetStringSize(myParam: String as LPStr): Integer  library
"TurboDBStudioDemoDll.dll";
dllproc StrEqual(Param1: String; Param2: String as LPStr): Integer  library
"TurboDBStudioDemoDll.dll";

dllproc WriteMemoToFile(Memo: String as LPStr) library
"TurboDBStudioDemoDll.dll";



procedure Hallo
  HelloWorld
endproc

procedure IntegerResult
  Message(Str(GetIntResult), "Sollte 12345 sein");
endproc

procedure IntegerGetTest;
  Message(Str(GetIntParameter(39)));
endproc

procedure RealGetTest;
  Message(Str(GetRealParameter(3.9),5,2));
endproc

procedure RealSetTest;
  vardef x: Real;
  x := 10.5;
  SetRealParameter(x);
  Message("Aufruf liefert " + Str(x,5,2) + ". Erwartet werden 21,00.");
endproc

procedure IntegerlSetTest;
  vardef x: Integer;
  x := 10;
  SetIntParameter(x);
  Message("Aufruf liefert " + Str(x,5,2) + ". Erwartet werden 20,00.");
endproc

procedure ShowEmpty(s: String);
  if IsStringEmpty(s)
    Message("Der String ist leer")
  else
```

```
    Message("Der String lautet: " +chr(13)+ s)
  end;
endproc

procedure ShowEqual
  vardef res: real;
  res := StrEqual("aSdDfFgH", "AsDdFfGh")
  if res = 0
    Message("Der String ist gleich", Str(res))
  else
    Message("Der String ist ungleich", Str(res))
  end;
endproc

procedure StringTest
  vardef s: String;
  s := ""
  ShowEmpty(s)
  s := "dhfgksldfhglsjkdhfglskdhfglks hdlfkghsdjkfhgskdlfhk"
  SetTurboPLString(s, 25);
  ShowEmpty(s)
endproc

procedure MemoToFile;
  WriteMemoToFile(Tabelle.MemoFeld);
endproc
```

4.3.1.8 EC Steuerkommando

Syntax

```
.EC X
```

Erläuterung

Schaltet die Fehlerbehandlung durch den Anwender ein und aus. Wenn EC auf 1 gesetzt ist, wird bei einem Fehler in eine Prozeduraufruf nicht abgebrochen, wenn der Funktionswert des Aufrufs in einer Variablen gespeichert wird. In diesem Fall geht die Ausführung des Programms weiter und der Fehlerzustand steht in der Fehlervariablen *Error*.

Wert von X	Bedeutung
0	Abbruch des Datenbankjobs bei Fehlern. Voreinstellung.
1	Kein Abbruch, Variable "Fehler" enthält Fehlerstatus

Verwendung

In Datenbankjobs und Module. Meist zu Beginn der Datei.

Beispiel

Routine für die Eingabe einer gültigen Zahl mit Input:

```
procedure ZahlEingabe: Real
   vardef Zahl: Real;
   .EC 1
   repeat
      Input("Geben Sie eine Zahl ein", );
      Zahl := Val(T-Eingabe);
   until Error.Nummer = 0
   .EC 0
   return Zahl
endproc
```

4.3.1.9 end/ende Kommando

Erklärung

Abschluss einer if, while- oder repeat-Anweisung

Beispiel

Siehe if

4.3.1.10 ExecProg Prozedur

Syntax

```
ExecProg(Dateiname: String; [Oem: Integer]): Integer
ExecProg(Feld: String[]): Integer
ExecProg(Memo: Memofeld): Integer
```

Erklärung

ExecProg gibt es in drei verschiedenen Varianten. In jeder führt es eine Folge von TurboPL -Anweisungen aus. Diese Anweisungen stehen bei der ersten Variante in einer Datei, bei der zweiten in einem String-Array und in der dritten in einem Memo-Feld. Im ersten Fall gibt Oem an, ob es sich um DOS-Zeichensatz (OEM = 1) oder Windows-Zeichensatz (OEM = 0) handelt. Wenn das Argument ein String-Array ist, muss in jedem String eine TurboPL-Zeile stehen. Die Ausführung endet bei der ersten leeren Zeile. Das Ergebnis ist im Erfolgsfall Null und sonst ein Fehlercode.

Beispiel

Die Speicher-Datei wird mit Befehlen beschrieben. Diese werden dann in ein Memo gelesen und dann ausgeführt.

```
procedure START_EXT_PROGRAMM
    vardef nFI : REAL;
    nFI := Rewrite("RAMTEXT");
    WriteLn(nFI, "Message('Hallo')");
    Close(nFI);
    ReadMemo(KUNDEN.Memo, "RAMTEXT", 1);
    ExecProg(KUNDEN.Memo);
endproc
```

4.3.1.11 Execute Prozedur

Syntax

```
Execute(FileName: String; Wait: Integer [; Mode: Integer]): Integer
Execute(Dateiname: String; Warten: Integer [; Modus: Integer]): Integer
```

Erklärung

Startet das Programm mit dem Dateinamen Dateiname. Komandozeilenparameter können dabei ebenfalls übergeben werden. Mit dem Parameter Warten kann eingestellt werden, ob *TurboDB Studio* auf die Beendigung des aufgerufenen Programmes warten soll. Im Erfolgsfall ist der Rückgabewert 0.

Mit *Execute* können auch Dokumente geöffnet werden, wenn man als *FileName* seinen Dateinamen angibt.

Warten	0	*TurboDB Studio* läuft weiter, während die Anwendung ausgeführt wird.
	1	*TurboDB Studio* ist inaktiv, während die Anwendung läuft.
Modus	-1	Die Anwendung wird unsichtbar gestartet.
	0	Die Anwendung wird normal gestartet (Vorgabe).
	1	Die Anwendung wird bildschirmfüllend gestartet.

Beispiele

Dieser Aufruf startet die Anwendung Paint, um das angegebene Bild zu bearbeiten:

```
Execute('pbrush.exe c:\vdp\beispiel\logo.bmp', 0);
```

Dieser Aufruf startet den Standard-Browser maximiert, um die HTML-Seite anzuzeigen und wartet, bis der Anwender den Browser wieder schließt:

```
Execute('c:\temp\index.html', 1, 1);
```

Siehe auch

IsTask

4.3.1.12 for..to... Kommando

Syntax

```
for <Integer-Variable> := <Startwert> to <Endwert>
  <Befehle>
next;
```

Erklärung

Die Integer-Variable wird zu Beginn der Schleife auf den Startwert gesetzt. Dann wird geprüft, ob der Endwert schon überschritten ist und wenn nicht, werden die Befehle ausgeführt. Wenn der next-Befehl erreicht ist, wird die Variable um 1 hochgezählt und wieder bei der Prüfung begonnen.

Beispiel

Die folgende Prozedur füllt ein Array mit fortlaufenden Werten von 1 bis zur Obergrenze des Arrays:

```
procedure FillArray(var A: Real[])
  vardef i: Integer;
  for i := 1 to High(1, A)
    A[i] := i;
  next;
endproc;
```

Siehe auch

While, Repeat

4.3.1.13 GetEnv Prozedur

Syntax

```
GetEnv(Umgebungsvariable: String): String
```

Erkärung

Liefert den Wert einer Umgebungsvariablen, falls diese definiert ist, andernfalls einen Leerstring.

Beispiel

```
Execute(GetEnv("COMSPEC"),"")
GetEnv("WINDIR")
```

Das erste Beispiel führt die DOS-Shell aus, das zweite liefert das aktuelle Windows-Verzeichnis.

4.3.1.14 GetProductId Prozedur

Syntax

```
GetProductId: String
GibProduktId: String
```

Erklärung

Diese Funktion ermittelt die VDP-Lizenznummer, bzw die Lizenznummer der Anwendung.

Beispiel

```
Message(GetProductId, "Lizenznummer");
```

4.3.1.15 Halt Prozedur

Syntax

```
Halt
```

Erklärung

Die Funktion *Halt* beendet die Ausführung eines Makros oder eines Datenbankjobs mit sofortiger Wirkung.

Beispiel

```
if UserName <> "Huber"
   Message('User ' + UserName + darf diese Aktion nicht durchführen.');
   Halt;
end
```

Siehe auch

Return, Exit

4.3.1.16 if Kommando

Syntax

```
if Selektion
   Kommandos
end
```

Erklärung

Bedingte Bearbeitung von Kommandos

Hier werden die Kommandos nur dann bearbeitet, wenn die Selektion zutrifft. Andernfalls wird mit dem nächsten Kommando nach *END* fortgefahren.

```
if Selektion
   Kommandos1
else
   Kommandos2
end
```

Auch hier werden die Kommandos1 ausgeführt, wenn die Selektion zutrifft. Im anderen Fall jedoch werden die Kommandos2 ausgeführt. Die Selektion hat hier die Funktion eines binären Schalters.

Es können jedoch noch mehr Alternativen auftreten. Dafür kann man jeweils einen eigenen *ELSIF* -Zweig einbauen:

```
if Selektion1
   Kommandos1
elsif Selektion2
   Kommandos2
elsif Selektion3
   Kommandos3
...
else
   KommandosX
end
```

Hier wird zunächst die erste Selektion geprüft. Trifft sie zu, so werden die *Kommandos1* ausgeführt und anschließen die Bearbeitung nach dem END fortgesetzt. Andernfalls wird *Selektion2* geprüft, und im positiven Fall werden die *Kommandos2* ausgeführt und dann ebenfalls an das Kommando nach dem *END* gesprungen. Das gleiche wiederholt sich für alle anderen Alternativen mit *ELSIF*, bis keine weitere mehr vorhanden ist. Nur in diesem Fall werden die *KommandosX* nach dem *ELSE* ausgeführt.

Statt *IF*, *ELSE* und *ELSIF* können auch die deutschen Entsprechungen *FALLS*, *SONST* und *ANDERNFALLS* benutzt werden.

Beispiel

Anrede im Datenbankjob:

```
.IF Geschlecht ist weiblich
   Sehr geehrte Frau $Name
.ELSE
   Sehr geehrter Herr $Name
.END
```

4.3.1.17 include Kommando

Syntax

```
include Textdatei
```

Kategorie

Datenbankjobs

Erklärung

Einfügen eines Textbausteins in einen Datenbankjob.

Dem Kommando *include* wird der Name einer Textdatei nachgestellt. An dieser Stelle wird die Bearbeitung des momentanen Jobs unterbrochen und erst die im Argument des Kommandos spezifizierte Textdatei vom Datenträger gelesen und komplett bearbeitet. Nach deren Bearbeitung wird der ursprüngliche Text wieder aufgenommen. Einfügeanweisungen dürfen auch geschachtelt werden, das heißt, auch der einzufügende Text darf wiederum eine (oder mehrere) Einfügeanweisungen enthalten.

Das Einsatzgebiet ist das Einfügen von Textbausteinen in Datenbankjobs. Ein per *include* eingebundener Text kann auch *TurboPL*-Quelltext enthalten.

Die eingefügten Textbausteine dürfen keine eigene Textaufteilung (Prolog etc.) enthalten.

In Kopf- und Fußbereich ist dieses Kommando nicht angebracht, da sonst kein korrekter Seitenumbruch durchgeführt werden kann.

Das *include*-Kommando verarbeitet nur Quelltext. Im Gegensatz zu *ExecProg* wird das Kommando *include* während des Kompilierens und nicht während der Laufzeit ausgeführt.

Im einem Modul ist statt *include* das Kommando *USES* zu verwenden.

Beispiel

```
include Anreden.txt
include Vorbel.txt
```

Siehe auch

ExecProg, Uses

4.3.1.18 IsTask Prozedur

Dieser Befehl aus früheren Versionen ist unter Windows nicht sinnvoll und wurde deshalb entfernt.

4.3.1.19 IsUndef

Syntax

```
IsUndef(Value): Integer
```

Erklärung

Die Funktion *IsUndef* prüft, ob das übergebene Argument *Value* undefiniert ist.

Beispiel

ARTIKEL.Betrag ist hier eine Spalte vom Typ Float mit der Option *Unterscheidung zwischen 0 und leer* (nullable).

```
if IsUndef(ARTIKEL.Betrag)
      Message('Für diesen Artikel wurde noch kein Betrag festgelegt');
end
```

4.3.1.20 Kommentare

Syntax

```
..<Text>
```

Erläuterung

Kommentare werden durch mindestens zwei aufeinanderfolgende Punkte gekennzeichnet. Sie dienen nur der Dokumentation und haben keinen Einfluß auf die Bearbeitung.

Beispiel

```
procedure TueNichts
```

```
.. Diese Prozedur enthält nur Kommentarzeilen
.. und stellt mit Sicherheit nichts Schlimmes an
endproc
```

4.3.1.21 NB Steuerkommando

Syntax

```
NB n
```

Erklärung

NB (no break) bestimmt, ob ein Programm per Tastatur (Strg + U) abgebrochen werden darf.

0	Abbruch zulassen (Standard)
1	Abbruch unterdrücken

4.3.1.22 NLoop Prozedur

Syntax

```
NLoop(Laufvariable, Obergrenze: Integer, Ausdruck1, Ausdruck2, ...)
```

Erklärung

NLoop ist eine schnelle Schleife, die vor allem für Array-Berechnungen eingesetzt wird.

Laufvariable wird zunächst auf 0 gesetzt. Dann wird Obergrenze berechnet. Ist diese kleiner 0, so wird die Funktion ohne weitere Aktivität sofort abgebrochen. Andernfalls werden die einzelnen Ausdrücke *Ausdruck1*, *Ausdruck2* usw. berechnet und anschließend die Laufvariable um 1 erhöht. Das wird solange fortgesetzt, bis der Wert der Laufvariablen die Obergrenze entweder erreicht oder überschritten hat. Das Ergebnis der Funktion ist wiederum die Obergrenze.

Beispiel

Das REAL-Array Vektor wird komplett gelöscht, d.h. auf Null gesetzt.

```
vardef Vektor: Integer[1000]
vardef Index: Integer
NLoop(Index, High(1, Vektor), Vektor[Index]:=0)
```

Der selbe Effekt kann auch mit einer *While*-Schleife erzielt werden. Die Variante mit *NLoop* ist jedoch kürzer zu schreiben und schneller in der Abarbeitung:

```
vardef Vektor: Integer[1000]
vardef Index: Integer
Index := 0;
while Index <= High(1, Vektor)
      Vektor[Index] := 0;
      Index := Index + 1;
end
```

Siehe auch

While, Repeat

4.3.1.23 Note Prozedur

Veraltet, nicht mehr verwenden! Stattdessen können ab TurboDB Studio normale globale Variable eingesetzt werden.

Syntax

```
Note(Zeichenkette: String)
```

oder

```
Note
```

Erklärung

Legt die Zeichenkette in einem internen Speicher ab. *Note* ohne Parameter liefert den Inhalt des internen Speichers. Damit können Berichte oder Datenbankjobs von Makros aus mit Informationen versorgt werden.

Beispiel

Es soll der aktuelle Datensatz der Tabelle KUNDEN gedruckt werden. Die Tabelle verfügt über

eine Spalte Laufende_Nummer vom Typ Auto-Nummer. Dazu wird in einem Modul folgende Prozedur definiert:

```
prozedur AktuellenDSDrucken
    Note(Str(KUNDEN.Laufende_Nummer))
    Drucken("KUNDEN.Bericht_Hinweis", 1)
endproz
```

Im Bericht Bericht_Hinweis der KUNDEN-Tabelle ist nun noch unter *Bearbeiten/Bericht-Eigenschaften/Daten-Definfition* die Selektion zu definieren:

```
Laufende_Nummer = Val(Note)
```

Nachdem das Modul übersetzt und der Bericht gespeichert sind, kann die Prozedur aus einem Datenfenster der Tabelle KUNDEN heraus über den Menüpunkt *Ausführen/Makro ausführen...* gestartet werden.

4.3.1.24 ParamStr Prozedur

Syntax

```
ParamStr(ParameterNr: Integer): String
```

Erklärung

Für *ParameterNr* = 0 liefert *ParamStr* den Namen der aktuellen Anwendung. Im normalen Betrieb also etwas in der Art: C:\MyApp\MyApp.exe.

Ansonsten ist für *ParameterNr* die Nummer des abzufragenden Kommandozeilenparameters anzugeben.

4.3.1.25 Sel Prozedur

Syntax

```
Sel(<Bedingung>): Integer
```

Erklärung

Sel wertet die Bedingung aus und liefert 1, wenn sie zutrifft, andernfalls 0.

Beispiel

```
Sel(LeftStr("Müller",1) = "M")   ->  1
Sel(Name <> Name)                    ->  0
```

4.3.1.26 Sleep Prozedur

Syntax

```
Sleep(Delay: Integer)
Pause(Dauer: Integer)
```

Erklärung

Das Programm legt eine Pause von ca. *Delay* hundertstel Sekunden ein. Während dieser Zeit wird die Kontrolle an andere Windows-Prozesse abgegeben, wenn mit *ShowWait* eine Meldung auf dem Bildschirm ausgegeben wird.

Beispiele

```
procedure PauseOhneMultitasking
    Pause(500)  ..etwa 5 Sekunden nichts...
endproc

procedure PauseMitMultitasking
    ShowWait("Jetzt kommt eine kleine Pause")
    Pause(500)  ..etwa 5 Sekunden mit aktivierter Taskumschaltung...
    HideWait
endproc
```

4.3.1.27 sub Kommando

Syntax

```
sub [Selektion]
    ..Anweisungen
endsub [Indexdefinition]
```

Erklärung

Die Anweisungen zwischen *sub* und *endsub* werden für alle mit dem aktuellen Datensatz der Primärtabelle verknüpften Datensätze ausgeführt.

Ein Subreport führt, wenn es sich um die Primärtabelle handelt (und keine Verschachtelung vorliegt) zu einer kompletten (eventuell durch einen Filter eingeschränkten) Bearbeitung der Tabelle. Handelt es sich nicht um die Primärtabelle oder um einen verschachtelten Subreport, so werden die zum aktuellen Datensatz gehörenden Sätze der im Subreport angesprochenen Tabellen bearbeitet. Innerhalb des Subreports wird die Primärtabelle vorübergehend auf die dort angesprochene Tabelle (falls es sich um eine handelt) bzw. die letzte bei der Relationsbildung (falls mehrere Tabellen vom Subreport betroffen sind) gesetzt.

Subreports dienen zur Bearbeitung aller an einen Datensatz der Primärdatei gekoppelten Datensätze einer Sekundärdatei.

Beispiel

Tabelle	Felder
KUNDEN	Name, Vorname
ARTIKEL	Bezeichnung, Einzelpreis
BESTELL	Bestelldatum, Kunde(L->KUNDEN), Bezahlt
POSTEN	Bestellung(L->BESTELL), Artikel(L->ARTIKEL), Menge

```
primtableis KUNDEN
..
..In der folgenden Schleife werden alle Einträge der Kunden-Tabelle bearbeitet
sub
    ..Zugriff auf KUNDEN
    ..Primärtabelle KUNDEN
endsub
..
..In dieser Schleife wird für jeden Kunden, die Liste seiner Bestellungen
bearbeitet
sub
    ..Zugriff auf KUNDEN
    ..Primärtabelle KUNDEN
    sub
       ..Zugriff auf BESTELL
       ..Primärtabelle BESTELL
    endsub
endsub
..
..Hier wird die Liste aller gültigen Kombinataion aus Kunden und Bestellungen
bearbeitet
..(also alle Bestellungen mit zugehörigem Kunden) und dann für jede Bestellung
alle zu-
..gehörigen Posten mit den passenden Artikeln
sub
    ..Zugriff auf KUNDEN,BESTELL
    ..Relationsbildung: KUNDEN - BESTELL
    ..Primärtabelle BESTELL
    sub
       ..Zugriff auf ARTIKEL, POSTEN
       ..Relationsbildung: BESTELL - POSTEN - ARTIKEL
       ..Primärtabelle ARTIKEL
    endsub
endsub
```

Jeder einzelne Subreport kann mit einer eigenen Selektion versehen werden. Diese Selektion wirkt dann nur auf diesen, nicht aber auf darin verschachtelte Subreports. Die in der Selektion angesprochenen Tabellen werden ebenfalls zur Bildung der Relation (Datensatzkombinationen)

herangezogen. Von denjenigen Tabellen, die ausschließlich in der Selektion angesprochen werden (und nicht mehr im Innern des Subreports), wird (falls überhaupt ein angekoppelter Satz vorhanden ist), nur eine Kombination gebildet.

Beispiel

```
primtableis KUNDEN
sub KUNDEN.PLZ von "20000" bis "49999"
    ..Zugriff auf KUNDEN
    ..Es werden nur Kunden aus dem selektierten PLZ-Bereich
    ..zu Verfügung gestellt.
endsub
sub
    ..Zugriff auf Kunden
    ..Alle Kunden werden zur Verfügung gestellt
    sub ARTIKEL.Einzelpeis > 1000
       ..Zugriff auf BESTELL
       ..Es werden diejenigen an KUNDEN gekoppelten Sätze aus
       ..BESTELL zur Verfügung gestellt, denen über POSTEN wenigstens
       ..ein Artikel mit der angegebenen Bedingung zugeordnet werden kann.
       ..Falls mehrere passende Artikel verknüpft sind, wird dennoch
       ..nur eine Bestellung ausgegeben (automatischer Schnitt).
    endsub
endsub
```

Falls sich ein Subreport auf nur eine Tabelle bezieht, kann eine Sortierung der zu bearbeiteten Datensätze erzwungen werden. Dazu wird das Sortierkriterium in Form einer Indexbeschreibung dem *endsub* nachgestellt.

Beispiel

Das folgende Programmfragment setzt zunächst einen Filter auf die Tabelle KUNDEN. Innerhalb des Subreports werden daraufhin die Datensätze in der dem Filter entsprechenden Indexreihenfolge erzeugt (hier Land,PLZ). Das nach ENDSUB angegebene Sortierkriterium sorgt dann dafür, dass die dem Subreport zugeführten Datensätze dieser Reihenfolge (Name,Vorname) entsprechen.

```
primtableis KUNDEN
setaccess LANDPLZ
filter D,20000/D,49999
sub
  ..Zugriff auf KUNDEN
endsub Name, Vorname
```

Siehe auch

Sub (Funktion)

4.3.1.28 SV Steuerkommando

Syntax

```
SV n
```

Erklärung

Wenn eingeschaltet (n = 1), werden Variablen bei Zuweisungen und anderen Operationen streng auf den korrekten Typ geprüft. Z.B. kann man dann nicht einer Integer-Variablen eine Real-Variable zuweisen oder einer Zeit-Variablen eine Integer-Variable. Dies gilt auch für Werte, die als Argumente an Prozeduren übergeben werden.

Die strenge Typprüfung kann aus Gründen der Rückwärtskompatibilität auch ausgeschaltet werden (n = 0), dies wird allerdings nicht empfohlen.

4.3.1.29 repeat Kommando

Syntax

```
repeat
    <Kommandos>
until <Bedingung>
```

Erklärung

In dem Fall, dass ein Programmteil mehrfach, mindestens jedoch einmal ausgeführt wird, setzt

man das REPEAT-Kommando ein. Zunächst werden die Kommandos ausgeführt. Dann wird die Selektion geprüft. Trifft sie zu, so wird mit dem nächsten auf REPEAT folgenden Kommando weitergemacht. Andernfalls beginnt die Schleife von vorne. Statt REPEAT und UNTIL kann auch WIEDERHOLE und BIS verwendet werden.

Beispiel

```
repeat
   okay := Message("Soll Datei überschrieben werden", "Achtung", 3);
until okay = 6
```

Siehe auch

while, for...to...

4.3.1.30 return Kommando

Syntax

```
return Ausdruck
```

Erklärung

Eine Prozedur kann mit dem Kommando *return* vorzeitig verlassen werden. Falls es sich um eine Funktionsprozedur handelt, wird dem *return* deren Ergebnis nachgestellt. Damit eine Funktionsprozedur einen definierten Wert liefert, muss sie mindestens ein *return*-Kommando enthalten.

Beispiel

```
procedure SchreibDaten
   ...
   if Message("Soll Datei überschreiben werden", "Ausgabe", 4) <> 6
     return
   end
   ...
endproc


procedure Zinsen(Kapital, Steuersatz, Tage: Real): Real
   return  Kapital*Steuersatz*Zeit/(365*100)
endproc


procedure Fakultät(X : Integer): Integer
   if X <= 1
     return 1
   else
     return X * Fakultät(X - 1)
   end
endproc
```

Siehe auch

Exit, Halt, EndProg

4.3.1.31 uses Kommando

Syntax

```
uses Modulname
```

Erklärung

Um in einem Modul Prozeduren eines anderen Moduls verwenden zu können, muss dieses zu Beginn mit *uses* eingebunden sein. Eingebundene Module müssen sich immer im Verzeichnis des aufrufenden Moduls befinden. Wenn z.B. die Prozedur Proc1 im Modul MODUL1 definiert ist und in MODUL2 verwendet werden soll, sieht das so aus:

Beispiel

```
..Hier beginnt das Module MODUL2
..Als erstes wird MODUL1 eingebunden
uses MODUL1

procedure Proc2
```

```
    ..Jetzt kann Proc1 aus MODUL1 aufgerufen werden
    Proc1;
    ..Hier kommen die sonstigen Befehle
endproc
```

Siehe auch

Include

4.3.1.32 Wahrheitswerte

Die folgenden Konstanten werden verwendet um logische Werte auszudrücken. Das TurboPL keine eigentlichen logischen Werte kennt, werden die Werte für wahr als 1 und die Werte für falsch als 0 gesetzt.

Name	Bedeutung	Wert
Ja	wahr	1
True	wahr	1
Nein	falsch	0
False	falsch	0

Beispiel

Dem Tabellenfeld *Umsatzsteuerpflichtig* wird ein Wert zugewiesen:

```
KUNDE.Umsatzsteuerpflichtig := Ja;
```

das Gegenteil:

```
KUNDE.Umsatzsteuerpflichtig := False;
```

Dies bewirkt dasselbe wie die folgende Anweisung ist aber besser verständlich:

```
KUNDE.Umsatzsteuerpflichtig := 0;
```

4.3.1.33 while Kommando

Syntax

```
while Bedingung
    <Kommandos>
end
```

Erklärung

Wenn ein Programmteil mehrfach, unter bestimmten Bedienungen aber auch überhaupt nicht ausgeführt werden soll, benötigt man eine sogenannte abweisende Schleife. Hier wird zunächst die Selektion geprüft. Trifft sie zu, so werden die Kommandos ausgeführt. Andernfalls wird die Arbeit mit dem Kommando nach dem *end* fortgesetzt. Nach der Ausführung der Kommandos beginnt die Prüfung wieder von vorne.

Beispiel

```
while not Eot(Text)
    ReadLn(Text)
end;
```

Siehe auch

repeat, for..to...

4.3.2 Systemvariablen

4.3.2.1 Systemvariablen

Bei den Systemvariablen ist die Schreibweise signifikant, d.h. die vorgegebene Groß/Kleinschreibung muss eingehalten werden.

Error	beschreibt den letzten aufgetretenen Fehler
Fehler (obsolet)	Code des zuletzt aufgetretenen Fehlers
G_alt	Gruppendefinition vor Gruppenwechsel
G_neu	Gruppendefinition nach Gruppenwechsel

heute	Systemdatum als Zeichenkette
jetzt	Systemzeit als Zeichenkette
Seite	Nummer der aktuellen Druckseite
TDB-Pfad	Verzeichnis des TurboDB Studio
T-Eingabe	Ergebnis der Funktion Input
Zeile	Nummer der aktuellen Druckzeile

4.3.2.2 Error

Erklärung

Innerhalb eines except-Blocks enthält das Error-Objekt die Nummer und die Beschreibung des Fehlers. Es enthält die drei Felder:

Nummer/Number	Enthält die Fehlernummer wie im Kapitel Fehlermeldungen beschreiben.

Beschreibung/Description	Enthält eine zusätzliche Information wie z.B. einen Dateinamen.

Meldung/Message	Enthält die gesamte Fehlermeldung, wie sie dem Benutzer angezeigt wird.

Hinweis

Das Fehlerobjekt *Error* löst die alte Fehlervariable Systemvariable *Fehler* ab. Diese wird nur noch aus Gründen der Rückwärtskompatibilität unterstützt und sollte nicht mehr verwendet werden.

Beispiel

Im folgenden Beispiel wird eine Datensatz geschrieben. Falls dabei ein Ausnahme auftritt, zeigt der Code eine Meldung mit dem entsprechenden Text an.

```
try
   PostRec(KUNDEN);
except
   Message(Error.Beschreibung);
end;
```

4.3.2.3 Fehler (obsolet)

Hinweis

Die Systemvariable *Fehler* wurde durch das Systemvariable *Error* ersetzt. Diese hat den Vorteil, dass nicht nur der Fehlercode sondern auch eine Beschreibung des Fehlers geliefert wird. Deshalb sollten Sie *Fehler* nicht mehr verwenden; die Variable wird in einer zukünftigen Version von TurboDB Studio nicht mehr unterstützt.

Erklärung

Diese Systemvariable enthält den Code des zuletzt aufgetretenen Fehlers, gilt nur nach *.EC 1*.

Siehe auch

Steuerbefehl EC, Fehlermeldungen

4.3.2.4 G_alt

Erklärung

Gruppendefinition vor Gruppenwechsel

Im Gegensatz zur Systemvariable *G_Neu* muss *G_alt* vor Verwendung deklariert werden (Bitte auf Schreibweise achten!) Findet *TurboDB Studio* eine Variable dieses Namens vor, wird hier automatisch der Wert des Gruppenmerkmals vor dem Gruppenwechsel gespeichert. Damit eignet sich *$G_alt* für die Ausgabe des Gruppenmerkmals am Ende einer Gruppe.

Beispiel

Zählen der Adressen in den einzelnen Postleitzahlbereichen und Ausgabe im Anschluß an jede Gruppe:

```
.REPORT
.PROLOGUE
.SETACCESS PLZ.IND
.GP 0
.VAR G_alt=LeftStr(PLZ,1)
.GROUP LeftStr(PLZ,1)
$('Postleitzahlbereich ab ' G_alt'0000: 'ZCount(PLZ)' Adressen')
.DATA
```

Ausgabe:

Postleitzahlbereich ab 10000: 12 Adressen

Postleitzahlbereich ab 20000: 102 Adressen

...usw.

Siehe auch

Group, GP, G_Neu

4.3.2.5 G_Neu

Erklärung

Die Variable stellt das Gruppenmerkmal zur Ausgabe im Gruppen-Bereich zur Verfügung. Sie ist nur dann vorhanden, wenn der Datenbankjob das Bereichskommando .GROUP enthält.

Beispiel

Liste der Adressen gruppiert über Ortschaften mit Ausgabe des Gruppenmerkmals zu Beginn jeder Gruppe:

```
.REPORT
.PROLOGUE
.SETACCESS PLZ.IND
.GROUP PLZ+' '+Ort
$('Ort: 'G_Neu)
.DATA
$(Vorname:20 Name:30 Straße:20)
```

Ausgabe:

Ort: 94032 Passau

Konstantin	Weiß	Faulerstr. 23
Laura	Siebenweiß	Gruenaustr.64

Ort: 80809 München

Sabine	Heuer	Nadistr.43

...usw.

Siehe auch

Group, GP, G_alt

4.3.2.6 heute

Erklärung

Systemdatum als Zeichenkette

Beispiel

Serienbrief als Datenbankjob:

```
.Daten
Autohaus Hopfinger
Karrenweg. 19
95673 Radstadt

$Vorname $Name
.PO 8
.MT 4
$Straße
$PLZ $Ort

Radstadt, den $heute, $jetzt


Sehr geehrte Damen und Herren,
      ... usw.
```

Siehe auch

Today, jetzt

4.3.2.7 jetzt

Erklärung

Systemzeit als Zeichenkette

Siehe auch

Now, heute

4.3.2.8 Seite

Erklärung

Die Standardvariable Seite enthält die aktuelle Seitenzahl. Mit dem Steuerkommando *PN* kann der Startwert beliebig gesetzt werden.

Beispiel

Ausgabe einer Kundenliste per Datenbankjob:

```
.REPORT
.FOOTER
$(NTimes('-', 80))
$('Seite ' + Seite:70)
.DATEN
$(Name:30 Vorname:30)
```

Um den selben Effekt in einem Bericht zu erzielen, wird einfach ein Formelfeld (Anzeigeelement Formel) im Fußbereich des Berichts plaziert und der Ausdruck *Str(Seite)* als Formel angegeben.

Siehe auch

Zeile

4.3.2.9 TDB-Pfad

Erklärung

Die systemweit verfügbare Stringvariable *TDBPfad* beinhaltet das Verzeichnis in dem sich die Programmdatei von TurboDB Studio beziehungsweise der damit erstellen Anwendung befindet. Die Variable kann nur gelesen werden.

4.3.2.10 T-Eingabe

Erklärung

Nicht mehr verwenden. Diese Variable ist aus Kompatibilitätsgründen noch vorhanden Frühere Versionen von *Input* und *ChooseFile* verwenden diese systemweite Variable vom Typ String. In neueren Versionen kann man bei *Input* und *ChooseFile* eine String-Variable für den vom Anwender eingetragenen Wert angeben.

Beispiel

Siehe Input und ChooseFile

4.3.2.11 Zeile

Erklärung

Nummer der aktuellen Druckzeile

Beispiel

Ausgabe einer Kundenliste per Datenbankjob, wobei die Kunden fortlaufend Nummeriert werden.

```
.REPORT
.DATEN
$(Zeile:5 + '. ' + Name:30 Vorname:30)
```

Siehe auch

Seite

4.3.3 Fehlersuche

4.3.3.1 Fehlerbehandlung

Dignostische Funktionen dienen dazu, die korrekte Programmausführung zu prüfen und zu beobachten.

Trace	Schreibt eine Meldung in die Debug-Ausgabe
try...except...finally	Kommando

4.3.3.2 Trace Prozedur

Syntax

```
Trace(InfoText: String)
```

Erklärung

Gibt einen beliebigen Text zu Debugging-Zwecken aus. In der Entwicklungsumgebung erscheint der Text im Hinweisfenster. In übersetzen Anwendungen wird die Ausgabe unterdrückt.

Beispiel

Die Prozedur schreibt vor und nach einer Berechnung einen Hinweis in das Logbuch.

```
procedure ShowSqr(a: Real)
   vardef b: Real;
   Trace("Berechnung des Quadrates der Zahl: " + Str(a))
   .. Quadrat wird mit Hilfe der Exponentialfunktion gebildet
   b := Exp(2*Log(a))
   Message("Das Quadrat von "+Str(a)+" lautet "+Str(b), "Quadrat einer Zahl
berechnen")
   Trace("Quadrat wurde berechnet und ausgegeben")
endProc;
```

Siehe auch

Message

4.3.3.3 try...except...finally Kommando

Syntax

```
try
    ..Programmcode
except
    ..Fehlerbehandlung
[finally]
    ..Code, der immer ausgeführt wird
end;
```

Erklärung

Wenn im Programmcode zwischen *try* und *except* ein Laufzeitfehler auftritt, wird dieser Programmblock sofort verlassen und der Code für die Fehlerbehandlung zwischen *except* und *end* angesprungen. In diesem Block kann man auf die Fehlervariable Error zugreifen, welche einen Fehlercode und eine Beschreibung enthält. Der Einsatz von *finally* ist optional. Der zwischen *finally* und *end* definierte Code wird unabhängig davon ausgeführt, ob nach *try* ein Fehler aufgetreten ist und mit *except* abgefangen wurde. Hier stehen also Befehle, die unter keinen Umständen ausgelassen werden dürfen. Typischerweise werden hier gesetzte Sperren aufgehoben (siehe Beispiel zu *EditOn*).

Beispiel

Hier wird eine Zahl durch eine andere dividiert, die eventuell auch 0 sein könnte. Um diesen Fall abzufangen, wird eine spezielle Fehlerbehandlung definiert. Weitere Fehler, die beim Lesen oder Schreiben in die Tabelle auftreten können, werden gemeldet und die Prozedur dann abgebrochen.

```
vardef a, b: Integer;
try
    a := TABELLE.Feld1;
    b := TABELLE.Feld2;
    a := a div b;
    EditRec(TABELLE);
    TABELLE.Feld1 := a;
    PostRec(TABELLE);
except
    if Error.Nummer = 178
       ..Division durch Null
       a := 0;
    else
       Message(Error.Meldung)
       exit;
    end;
end;
```

4.3.4 Mathematik

4.3.4.1 Mathematik

Mit den Funktionen dieser Kategorie lassen sich auch komplexe Berechungen durchführen.

Abs	Berechnet den Absolutwert
Arctan	Berechnet den Arcus-Tangens
BitAnd	Berechnet die logische und-Verknüpfung zwischen zwei ganzen Zahlen oder Bit-Feldern
BitAndNot	Berechnet die logischen und-nicht-Verknüpfung zwischen zwei ganzen Zahlen oder Bit-Feldern
BitOr	Berechnet die logische oder-Verknüfpung zwischen zwei ganzen Zahlen oder Bit-Feldern
Cos	Berechnet den Kosinus einer Zahl
div	Führt eine ganzahlige Division durch
Exp	Berechnet die Exponentialfunktion einer Zahl
Frac	Berechnet den Nachkomma-Anteil einer Dezimalzahl
Int	Berechnet den ganzzahligen Anteil einer Dezimalzahl

Log	Berechnet den natürliche Logarithmus
mod	Berechnet den Rest bei ganzzahliger Division (Modulo)
Random	Bestimmt eine Zufallszahl
Round	Rundet eine Zahl
Sin	Berechnet den Sinus einer Zahl
Sqrt	Berechnet die Quadratwurzel einer Zahl
TestBit	Prüft, ob ein Bit in einer Zahl gesetzt ist

4.3.4.2 Abs Prozedur

Syntax

```
Abs(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Abs berechnet den Absolutbetrag eines nummerischen Ausdrucks. Falls der Wert der Zahl negativ ist, wird das Vorzeichen umgedreht. Man erhält also immer ein positives Ergebnis.

Laufzeitfehler

Keine

Beispiel

Ein häufiges Einsatzgebiet für diese Funktion ist der Vergleich eines Festkomma-Feldes mit einem berechneten Wert. Da solche Zahlen intern im Binärformat verarbeitet werden, können Rundungsunterschiede auftreten, weil die meisten Dezimalbrüche im Dualsystem nicht endlich dargestellt werden können. Aus diesem Grund ist nicht sichergestellt, dass 0.1=1/10. In einem solchen Fall prüft man nicht auf absolute Gleichheit, sondern fordert nur, dass sich beide Zahlen maximal um einen sehr kleinen Betrag unterscheiden:

```
Abs(0.1 - 1/10) < 0.00001
Abs(5)     --> 5
Abs(-5)    --> 5
Abs(5.5)   --> 5
```

Siehe auch

Cos, div, Exp, Frac, Int, Log, mod, Round, Sin, Sqrt

4.3.4.3 Arctan Prozedur

Syntax

```
Arctan(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Arctan berechnet den Arkustangens eines nummerischen Ausdrucks.

Laufzeitfehler

Keine

Siehe auch

Cos, div, Exp, Frac, Int, Log, mod, Round, Sin, Sqrt

4.3.4.4 BitAnd Prozedur

Syntax

```
BitAnd(Arg1: Integer; Arg2: Integer): Integer
BitAnd(var Arg1: Bit[]; Arg2: Bit[])
```

Kategorie

Basisfunktion

Erklärung

BitAnd berechnet die logische und-Verknüpfung aus *Arg1* und *Arg2*. Es gibt zwei Varianten:

- Die ersten Variante liefert die bitweise Verknüpfung von zwei Integer-Werten als Funktionsergebnis. Um das Ergebnis zu verstehen, muss man an die binäre Darstellung der Zahlen denken.
- Die zweite Variante operiert auf Bit-Arrays und speichert das Ergebnis in *Arg1*. D.h. nach dieser Operation, sind alle diejenigen Bits in *Arg1* gesetzt, die vorher sowohl in *Arg1* als auch in *Arg2* gesetzt waren.

Die logische und-Verknüpfung hat folgende Wahrheitstafel:

```
0 und 0 -> 0
1 und 0 -> 0
0 und 1 -> 0
1 und 1 -> 1
```

Laufzeitfehler

Keine

Beispiele

```
BitAnd(1, 1) -> 1 (Binäre Darstellung ist 0001 und 0001)
BitAnd(3, 1) -> 1 (Binäre Darstellung ist 0011 und 0001)
BitAnd(7, 3) -> 3 (Binäre Darstellung ist 0111 und 0011)
BitAnd(4, 3) -> 0 (Binäre Darstellung ist 0100 und 0011)
```

Siehe auch

BitNot, BitOr, BitXor, BitAndNot

4.3.4.5 BitAndNot Prozedur

Syntax

```
BitAndNot(Arg1: Integer; Arg2: Integer): Integer
BitAndNot(var Arg1: Bit[]; Arg2: Bit[])
```

Kategorie

Basisfunktion

Erklärung

BitAndNot berechnet die logische und-nicht-Verknüpfung aus *Arg1* und *Arg2*. Es gibt zwei Varianten:

- Die ersten Variante liefert die bitweise Verknüpfung von zwei Integer-Werten als Funktionsergebnis. Um das Ergebnis zu verstehen, muss man an die binäre Darstellung der Zahlen denken.
- Die zweite Variante operiert auf Bit-Arrays und speichert das Ergebnis in *Arg1*. D.h. nach dieser Operation, sind alle diejenigen Bits in *Arg1* gesetzt, die vorher zwar in *Arg1* aber nicht in *Arg2* gesetzt waren.

Die logische und-nicht-Verknüpfung hat folgende Wahrheitstafel:

```
0 und 0 -> 0
1 und 0 -> 1
0 und 1 -> 0
1 und 1 -> 0
```

Laufzeitfehler

Keine

Beispiele

```
BitAndNot(1, 1) -> 0 (Binäre Darstellung 0001 und 0001)
BitAndNot(1, 0) -> 1 (Binäre Darstellung 0001 und 0000)
BitAndNot(7, 3) -> 4 (Binäre Darstellung 0111 und 0011, 0100 entspricht 4)
BitAndNot(7, 4) -> 3 (Binäre Darstellung 0111 und 0100, 0011 enspricht 3)
```

Siehe auch

BitNot, BitAnd, BitAndNot, BitOr

4.3.4.6 BitOr Prozedur

Syntax

```
BitOr(Arg1: Integer; Arg2: Integer): Integer
BitOr(var Arg1: Bit[]; Arg2: Bit[])
```

Kategorie

Basisfunktion

Erklärung

BitOr berechnet die logische oder-Verknüpfung aus *Arg1* und *Arg2*. Es gibt zwei Varianten:

- Die ersten Variante liefert die bitweise Verknüpfung von zwei Integer-Werten als Funktionsergebnis. Um das Ergebnis zu verstehen, muss man an die binäre Darstellung der Zahlen denken.
- Die zweite Variante operiert auf Bit-Arrays und speichert das Ergebnis in *Arg1*. D.h. nach dieser Operation, sind alle diejenigen Bits in *Arg1* gesetzt, die vorher in *Arg1* oder in *Arg2* (oder in beiden) gesetzt waren.

Die Oder-Verknüpfung hat folgende Wahrheitstafel:

```
0 und 0 -> 0
1 und 0 -> 1
0 und 1 -> 1
1 und 1 -> 1
```

Laufzeitfehler

Keine

Beispiele

```
BitOr(1, 1) -> 1 (Binäre Darstellung ist 0001 und 0001)
BitOr(2, 1) -> 3 (Binäre Darstellung ist 0010 und 0001, 0011 entspricht 3)
BitOr(7, 3) -> 7 (Binäre Darstellung ist 0111 und 0011)
```

Siehe auch

BitNot, BitAnd, BitAndNot, BitXor

4.3.4.7 BitNot Prozedur

Syntax

```
BitNot(O: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

Invertiert die Bits aus dem Argument.

Laufzeitfehler

Keine

Beispiel

```
BitNot(0) -> -1   (Bitmuster: 0 = 0000, -1 = 1111)
BitNot(1) -> -2   (Bitmuster: 1 = 0001, -2 = 1110)
BitNot(-257) -> 256      (Bitmuster: -257 1110 1111, 256 = 0001 0000)
```

Anmerkung: Integer-Zahlen haben in TurboPL 64 Bits. Hier sind jeweils nur die unteren vier bzw.

acht angegeben.

Siehe auch

BitAnd, BitAndNot, BitOr, BitXor

4.3.4.8 BitShl Prozedur

Syntax

```
BitShl(O: Integer[; N: Integer]): Integer
```

Kategorie

Basisfunktion

Erklärung

Verschiebt die Bits von O um N Positionen nach links. Falls N nicht angegeben ist, wird 1 angenommen.

Beispiele

```
BitShl(1, 1) -> 2 (Bitmuster: 1 = 0001)
BitShl(3, 2) -> 12(Bitmuster: 3 = 0011, 12 = 1100)
```

Siehe auch

BitShr

4.3.4.9 BitShr Prozedur

Syntax

```
BitShr(O: Integer[; N: Integer]): Integer
```

Kategorie

Basisfunktion

Erklärung

Verschiebt die Bits von O um N Positionen nach rechts. Falls N nicht angegeben ist, wird 1 angenommen.

Beispiele

```
BitShr(1, 1) -> 0 (Bitmuster: 1 = 0001, 0 = 0000)
BitShr(2, 1) -> 1 (Bitmuster: 2 = 0010, 1 = 0001)
BitShr(7, 2) -> 1 (Bitmuster: 7 = 0111, 2 = 0001)
```

Siehe auch

BitShl

4.3.4.10 BitXor Porzedur

Syntax

```
BitXor(Arg1: Integer; Arg2: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

BitXor berechnet die bitweise exklusive oder-Verknüpfung aus *Arg1* und *Arg2*. Exklusiv bedeutet, dass nur genau eines der beiden beteiligten Bits gesetzt sein darf. Das ist auch der Unterschied zur normalen Oder-Verknüpfung, wo das Ergebnis auch dann 1 ist, wenn beide Argumente 1 sind.

Die exklusive Oder-Verknüpfung hat folgende Wahrheitstafel:

```
0 und 0 -> 0
1 und 0 -> 1
0 und 1 -> 1
1 und 1 -> 0  ..Hier ist der Unterschied zu BitOr
```

Laufzeitfehler

Keine

Beispiele

```
BitXor(1, 1) -> 0 (Binäre Darstellung ist 0001 und 0001)
BitXor(2, 1) -> 3 (Binäre Darstellung ist 0010 und 0001, 0011 entspricht 3)
BitXor(7, 3) -> 4 (Binäre Darstellung ist 0111 und 0011, 0100 entspricht 4)
```

Siehe auch

BitAnd, BitAndNot, BitOr

4.3.4.11 Cos Prozedur

Syntax

```
Cos(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Liefert den Cosinus einer Zahl.

Laufzeitfehler

Keine

Beispiel

```
PI := 3.1415926536
Cos(PI/3)  ->  0.5
```

Siehe auch

Abs, div, Exp, Frac, Int, Log, Round, mod, Sin, Sqrt, Arctan

4.3.4.12 div Operator

Syntax

```
Zahl1 div Zahl2
```

Kategorie

Basisfunktion

Erklärung

Berechnet, wie oft *Zahl2* in *Zahl1* paßt (ganzzahlige Division). Der dabei entstehende Rest wird durch die Funktion mod ermittelt. Sowohl *Zahl1* als auch *Zahl2* müssen ganze Zahlen (Integer) sein.

Laufzeitfehler

178 Zahl2 ist Null.

Beispiel

```
A := B div 4;
```

Resultat:

B:	0	1	2	3	4	5	6
A:	0	0	0	0	1	1	1

Siehe auch

mod, Abs, Cos, Exp, Frac, Int, Log, Round, Sin, Sqrt

4.3.4.13 Exp Prozedur

Syntax

```
Exp(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Liefert die Exponentialfunktion der übergebenen Zahl. Es gilt *Log(Exp(Zahl)) = Exp(Log(Zahl)) = Zahl*.

Zusammen mit Log kann jede Potenzierung vorgenommen werden. Die Formel hierfür lautet: *X hoch Y = Exp(Y*Log(X))*

Laufzeitfehler

43 Das Ergebnis wird zu groß.

Beispiel

```
Exp(0)        ->   1
Exp(1)        ->   2.7182818285
Exp(4*Log(2)) ->   16
```

Siehe auch

Abs, Cos, div, Frac, Int, Log, mod, Round, Sin, Sqrt

4.3.4.14 Frac Prozedur

Syntax

```
Frac(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Ermittelt den gebrochenzahligen Anteil der Zahl. Ist *Frac(Zahl)=0*, so handelt es sich um eine ganze Zahl.

Laufzeitfehler

Keine

Beispiel

```
Frac(12.5)    ->  0.5
Frac(-12.5)   ->  0.5
Frac(10/3)    ->  0.33333333333
if Frac(Startwert)<>0
   Message("Nur ganze Zahlen eingeben","Achtung",1)
end
```

Siehe auch

Abs, Cos, div, Exp, Int, Log, mod, Round, Sin, Sqrt

4.3.4.15 Int Prozedur

Syntax

```
Int(Zahl: Real): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert den ganzzahligen Anteil einer Zahl. Dabei wird auch das Vorzeichen berücksichtigt. Es gilt

```
Int(Zahl) + Frac(Zahl) = Zahl
```

Laufzeitfehler

Keine

Beispiel

```
Int(12.5)        ->  12
Int(-12.5)       -> -12
Int(10/3)        ->   3
```

Siehe auch

Abs, Cos, div, Exp, Frac, Log, mod, Round, Sin, Sqrt

4.3.4.16 Log Prozedur

Syntax

```
Log(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Berechnet den natürlichen Logarithmus der Zahl. Es gilt

```
Log(Exp(Zahl)) = Exp(Log(Zahl)) = Zahl
```

Zusammen mit *Exp* kann jede Potenzierung vorgenommen werden. Die Formel hierfür lautet:

```
X hoch Y  = Exp(Y*Log(X))
```

Laufzeitfehler

43 Wenn das Argument negativ ist.

Beispiel

```
Log(1)              ->  0
Log(2.7182818285)   ->  1
```

Siehe auch

Abs, Cos, div, Exp, Frac, Int, mod, Round, Sin, Sqrt

4.3.4.17 mod Operator

Syntax

```
Zahl1 mod Zahl2
```

Kategorie

Basisfunktion

Erklärung

Liefert den Rest der ganzzahligen Division von Zahl1 durch Zahl2 (modulo).

Der Zusammenhang zwischen div und *mod* lautet:

```
(A div B)*B = A - (A mod B)
```

Laufzeitfehler

178 Wenn *Zahl2* Null ist.

Beispiel

```
A := B mod 4;
```

Resultate:

B:	0	1	2	3	4	5	6
A:	0	1	2	3	0	1	2

Siehe auch

Abs, Cos, div, Exp, Frac, Int, Log, Round, Sin, Sqrt

4.3.4.18 Random Prozedur

Syntax

```
Random(Zahl: Integer): Integer;
```

Kategorie

Basisfunktion

Erklärung

Liefert eine Zufallszahl zwischen 0 und Zahl - 1.

Laufzeitfehler

Keine

Beispiel

Eine Tabelle wird stichprobenartig geprüft, wobei in etwa jeder hundertste Eintrag untersucht werden soll.

```
.REPORT
.DATA
.IF Random(100)=0
..Prüfung durchführen
.END
.EPILOGUE
```

Eine Prozedur soll eine Zufallszahl zwischen eins und sechs für ein Würfelspiel ermitteln:

```
procedure Würfel: Integer;
   return Random(6) + 1;
endproc
```

4.3.4.19 Round Prozedur

Syntax

```
Round(Zahl: Real): Integer
Round(Zahl: Real; Nachkommastellen: Integer): Real
```

Kategorie

Basisfunktion

Erklärung

Rundet die angegebenen Zahl auf eine bestimmt Zahl an Nachkommastellen. In der ersten Fassung ist das Ergebnis eine ganze Zahl, in der zweiten eine Fließkommazahl. Der Unterschied zwischen *Round(2.3)* und *Round(2.3, 0)* besteht darin, in welchem Format das Ergebnis abgespeichert wird.

Laufzeitfehler

Keine

Beispiel

Funktion	Ergebnis
Round(100.234567)	100
Round(100.234567, 0)	100.0
Round(100.234567, 1)	100.2
Round(3.14159, 3)	3.142

Siehe auch

Abs, Cos, div, Exp, Frac, Int, Log, mod, Sin, Sqrt

4.3.4.20 Sin Prozedur

Syntax

```
Sin(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Berechnet den Sinus einer Zahl.

Laufzeitfehler

Keine

Beispiel

```
PI:=3.1415926536
Sin(PI/6)   ->   0.5
```

Siehe auch

Abs, Cos, div, Exp, Frac, Int, Log, mod, Round, Sqrt

4.3.4.21 Sqrt Prozedur

Syntax

```
Sqrt(Zahl: Real): Real
```

Kategorie

Basisfunktion

Erklärung

Berechnet die Quadratwurzel einer Zahl.

Laufzeitfehler

43	Wenn das Argument negativ ist.

Beispiel

```
Sqrt(16)  ->  4
```

Siehe auch

Abs, Cos, div, Exp, Frac, Int, Log, mod, Round, Sin

4.3.4.22 TestBit Prozedur

Syntax

```
TestBit(X, BitPos: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

Mit Hilfe der Funktion TestBit kann überprüft werden, ob ein bestimmtes Bit gesetzt ist oder nicht. Der Rückgabewert ist entsprechend entweder 1 oder 0.

X	beliebige ganze Zahl
BitPos	Bitposition beginnend bei 0 für das niederwertigste Bit

Laufzeitfehler

Keine

Beispiele

```
TestBit(1, 0) -> 1
TestBit(2, 0) -> 0
TestBit(7, 0) -> 1
TestBit(7, 1) -> 1
TestBit(7, 2) -> 1
TestBit(7, 3) -> 0
```

Besonders wichtig ist die Funktion im Zusammenspiel mit CommState zur Prüfung des seriellen Kommunikationskanals:

```
procedure DateiEnde(Kanal: Integer): Integer;
    vardef Result: Integer
    if TestBit(CommState(Kanal), 5)
       Result := 1;
    else
       Result := 0;
    end
    return Result;
endproc;
```

Siehe auch

CommState

4.3.5 Zeichenketten

4.3.5.1 Zeichenketten

Beim Arbeiten mit Zeichenketten, stehen sowohl Operatoren als auch Funktionen zur Verfügung.

Operatoren

+ Addiert zwei Zeichenketten: *S := S1 + S2*
=, <, >, <=, >= Vergleicht zwei Zeichenketten unter Berücksichtigung von Groß- und Kleinschreibung: *if S1 <= S2*
like/wie Vergleicht zwei Zeichenketten ohne Berücksichtung von Groß- und Kleinschreibung und erlaubt Platzhalter * und ?: *if S1 wie 'Mül*'*
contains/enthält Prüft, ob ein String in einem anderen enthalten ist: *if S1 enthält 'ü'*
[p] Greift auf das p-te Zeichen im String zu: *ZweitesZeichen := S[2]*
[p,n] Holt die Teilzeichenkette der Länge n ab dem Zeichen an Position p: *TeilString := S[3, 5]*

Funktionen

AnsiToOem	String von Windows- in DOS-Zeichensatz konvertieren
Asc	ANSI-Code des Zeichens
Chr	Zeichenkette aus einem ANSI-Code berechnen
DigitStr	Textdarstellung der einzelnen Ziffern berechnen, z.B. drei * fünf * neun.
Exchange	Ersetzt Teilstrings im String durch andere Strings
FillStr	Zeichenkette mit einem Zeichen befüllen
LeftStr	Linken Teil einer Zeichenkette berechnen.
Length	Länge einer Zeichenkette ermitteln
Lower	Zeichenkette in Kleinbuchstaben umwandeln
LTrim	Linke Leerzeichen entfernen
MemoStr	Kopiert die ersten Zeichen eines Memos in eine String-Variable.
Memo2HTML	Wandelt den Inhalt des Memos in eine HTML-Seite um
NewGuid	Erzeugt einen neuen GUID-Wert
NTimes	Zeichenkette aus Vervielfachung einer Zeichenkette
OemToAnsi	String von DOS- in Windows-Zeichensatz konvertieren
Pos	Position der kleinen Zeichenkette innerhalb der großen Zeichenkette
RightStr	Den rechten Teil einer Zeichenkette bestimmen
RTrim	Rechte Leerzeichen entfernen
Scan	Zählt, wie oft eine Zeichenkette in einer anderen vorkommt.
SoundEx	Umwandlung in phonetischen Code
Str	Zahl in Zeichenkette umwandeln
StrAdd	Zahl1 und Zahl2 als Zeichenketten addieren und Ergebnis als Zeichenkette zurückliefern.
StrComp	Fehlertoleranter String-Vergleich
RealVal	Konvertiert eine Zeichenkette in eine Zahl
Swap	Liefert Indexausdruck, wenn im Zeichenkettenausdruck das ^-Zeichen

	vorkommt.
TestLn	Zeilenumbruch und Vorschub
ToHtml	Zeichenkette von ANSI nach HTML konvertieren
Upper	In Großschreibung umwandeln
Val	Nummerischen Wert einer Zeichenkette ermitteln
ValStr	Wert einer Zeichenkette ermitteln

4.3.5.2 AnsiToOem Prozedur

Syntax

```
AnsiToOem(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt die Zeichenkette vom ANSI- (Windows-) Zeichensatz in den OEM- (DOS-) Zeichensatz um und liefert dies als Ergebnis. Die Funktion wird benötigt, wenn Sie Text aus *TurboDB Studio* für eine DOS-Anwendung aufbereiten wollen.

Laufzeitfehler

Keine

Beispiel

Hier wir eine Zeichenkette aus dem Feld Name einer TurboDB-Tabelle gelesen und in eine Datei geschrieben, die dann von einer DOS-Applikation gelesen werden kann:

```
VARDEF h: Integer;
h := Rewrite("TDATEI.TXT");
WriteLn(h, AnsiToOem(Name));
Close(h);
```

Siehe auch

OemToAnsi, Text-Funktionen

4.3.5.3 Asc Prozedur

Syntax

```
Asc(Zeichenkette: String): Integer
```

Kategorie

Basisfunktion

Erklärung

ANSI-Code des ersten Zeichens der Zeichenkette.

Laufzeitfehler

Keine

Beispiel

```
Asc("A")                 -> 65
Asc("Aber")   -> 65
```

Siehe auch

Chr, Text-Funktionen

4.3.5.4 Chr Prozedur

Syntax

```
Chr(Zahl: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Die Funktion liefert eine Zeichenkette mit einem Zeichen. Dieses Zeichen hat den übergebenen ANSI-Code.

Laufzeitfehler

Keine

Beispiel

```
Chr(65) -> "A"
```

Siehe auch

Asc, Text-Funktionen

4.3.5.5 DigitStr Prozedur

Syntax

```
DigitStr(Zahl: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Die Funktion berechnet den Nummerischen Ausdruck, rundet ihn zur nächsten ganzen Zahl und übersetzt diese dann in eine Folge von Zahlwörtern. Die Zahlwörter werden durch einen Stern voneinander getrennt. Diese Art der Darstellung eignet sich beispielsweise für Scheckausdrucke. Diese Funktion heißt in der DOS-Version *ZStr*.

Laufzeitfehler

Keine

Beispiel

```
DigitStr(1234) -> "eins*zwei*drei*vier"
```

Siehe auch

Text-Funktionen

4.3.5.6 Exchange Prozedur

Syntax

```
Exchange(Zeichenkette1, Zeichenkette2, Zeichenkette3: String): String
```

Kategorie

Basisfunktion

Erklärung

In der *Zeichenkette1* werden alle Vorkommen von *Zeichenkette2* durch *Zeichenkette3* ersetzt. Ist identisch mit und ersetzt die Funktion Tausch aus der DOS-Version.

Laufzeitfehler

Keine

Beispiel

```
Anschrift := Exchange(Anschrift, "Str.", "Straße")
```

Siehe auch

Text-Funktionen, *Subst*

4.3.5.7 FillStr Prozedur

Syntax

```
FillStr(s1, s2: String; Length: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Die Funktion gibt eine Zeichenkette zurück, die zunächst aus der übergebenen Zeichenkette *s1* gebildet wird. Ist die Länge des daraus resultierenden Strings kleiner als *Length*, wird mit *s2* aufgefüllt.

Laufzeitfehler

Keine

Beispiel

```
FillStr('Peter', '.', 10)          ->         "Peter....."
FillStr('', 'Hu', 10)                    ->         "HuHuHuHuHu"
FillStr('Zu lange', '..', 2)       ->         "Zu"
```

Siehe auch

Text-Funktionen

4.3.5.8 has Operator

Syntax

```
<string> has <substring>
<Zeichenkette> enthält <Unterzeichenkette>
```

Kategorie

Basisfunktion

Laufzeitfehler

Keine

Erklärung

Der Ausdruck ist wahr, wenn <*Unterzeichenkette*> in <*Zeichenkette*> enthalten ist. In alten Versionen von *TurboDB* konnte auch hat statt enthält verwendet werden.

4.3.5.9 LeftStr Prozedur

Syntax

```
LeftStr(Str: String; Count: Integer): String
Links(Zeichenkette: String; Anzahl: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Liefert die ersten der Zahl entsprechenden Zeichen von der Zeichenkette. Falls mehr Zeichen als vorhanden abgetrennt werden, wird die komplette Zeichenkette zurückgeliefert.

Laufzeitfehler

Keine

Beispiel

```
LeftStr("Hans Huber",5)   ->  "Hans "
LeftStr("Hans Huber",100) ->  "Hans Huber"
LeftStr("Hans Huber",1)   ->  "H"
```

Siehe auch

Text-Funktionen

4.3.5.10 Length Prozedur

Syntax

```
Länge(Zeichenkette: String): Integer
Length(Str: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Ermittelt die Länge einer Zeichenkette.

Laufzeitfehler

Keine

Beispiel

```
Length("Hans"+"Huber")  ->  9
Length("Hans Huber")    -> 10
```

Hier werden alle Datensätze markiert, bei denen im Namenfeld mehr als 30 Zeichen belegt sind:

```
Link(Length(KUNDEN.Name) > 30, SetMark(KUNDEN, RecNo(KUNDEN)))
```

Siehe auch

Text-Funktionen

4.3.5.11 Lower Prozedur

Syntax

```
Lower(Zeichenkette: String): String
Klein(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt die Zeichenkette in Kleinschreibung. Alle Zeichen der Zeichenkette außerhalb des Buchstabenbereichs werden nicht umgewandelt. Das Zeichen "ß" wird ebenfalls nicht behandelt. Soll dieses Zeichen in "SS" umgewandelt werden, so kann das durch eine Kombination dieser Funktion mit *Exchange* erfolgen.

Diese Funktion ist damit das genaue Gegenstück zur Funktion *Upper*.

Laufzeitfehler

Keine

Beispiel

```
Lower("Das ist eine Überschrift")            -> "das ist eine überschrift"
Lower("Titel in Kleinbuchstaben")            -> "titel in kleinbuchstaben"
Exchange(Lower("Straße"),"ß","SS")           -> "strasse"
```

Siehe auch

Upper, Text-Funktionen

4.3.5.12 LTrim Prozedur

Syntax

```
LTrim(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Liefert die übergebene Zeichenkette, nachdem alle führenden Leerzeichen entfernt wurden.

Laufzeitfehler

Keine

Beispiel

```
LTrim("   Hans Huber") -> "Hans Huber"
```

Siehe auch

RTrim, Text-Funktionen

4.3.5.13 MemoStr Prozedur

Syntax

```
MemoStr(Memo: Field [; Length: Integer]): String
```

Kategorie

Basisfunktion

Erklärung

Liest die ersten Zeichen des Memos. Wenn *Length* angegeben ist, werden maximal so viele Zeichen gelesen. Wenn *Length* nicht angegeben ist, werden bis zu 255 Zeichen gelesen. Wenn *Length* 0 ist, wird das gesamte Memo zurückgeliefert.

Laufzeitfehler

Keine

Beispiel

```
vardef s: String;
s := MemoStr(KUNDEN.Bemerkung);
```

Siehe auch

CopyMemo, Text-Funktionen

4.3.5.14 Memo2HTML Prozedur

Syntax

```
Memo2HTML(Memo: Field; FileName, BaseURL, FirstPicture, DefaultPicture,
BackPicture: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt den Text eines Memofeldes in eine komplette HTML-Seite um und schreibt diese in eine Datei.

Memo	Memofeld
Dateiname	In dieses File wird die HTML-Seite geschrieben RAMTEXT geht nicht
BaseURL	Virtuelle URL der Seite, wird nicht benötigt
FirstPicture	Wird nicht benutzt
DefPic	Pfad zu einer Bilddatei
BackPic	Hintergrundbild

Laufzeitfehler

Keine

Siehe auch

Text-Funktionen

4.3.5.15 NewGuid Prozedur

Syntax

```
NewGuid: String
```

Kategorie

Basisfunktion

Beschreibung

Berechnet eine neue GUID (Globally Unique Identifier = Weltweit eindeutiger Bezeichner). GUIDs sind ein Standard-Mechanismus von Windows, um Elemente mit einem eindeutigen Namen zu versehen, auch wenn diese Namen auf verschiedenen Rechner zur selben Zeit generiert werden. Dazu wird unter anderem auch die Nummer der Netzwerkkarte verwendet, falls vorhanden. Eine GUID sieht zum Beispiel so aus: {8E9DCE93-B6DC-4D0A-87A5-3D038E214533} Sie eignet sich hervorragend, um Datensätze eindeutig zu kennzeichnen, wenn die selbe Tabelle in mehreren

Kopien auf verschiedenen Arbeitsplätzen bearbeitet werden soll und anscheinend wieder zusammengemischt werden muss. Zum Speichern von GUIDs verfügt *TurboDB* über einen eigenen Datentyp.

Laufzeitfehler

Keine

Beispiel

Hier wird einem GUID-Feld in einer Tabelle als Wert eine völlig neue GUID zugewiesen.

```
ReadRec(TABELLE, 0);
TABELLE.Guid := NewGuid;
WriteRec(TABELLE, FileSize(TABELLE)+1);
```

Siehe auch

Text-Funktionen

4.3.5.16 NTimes Prozedur

Syntax

```
NTimes(Zeichenkette: String; Zahl: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Diese Funktion wiederholt die übergebene Zeichenkette sooft, wie in Zahl angegeben wird. Das Resultat ist die so entstandene Zeichenkette. Ist identisch mit und ersetzt die DOS-Version *xMal*.

Laufzeitfehler

Keine

Beispiel

```
NTimes("*", 10)  -> "**********"
NTimes("ei", 5)  -> "eieieieiei"
```

Siehe auch

Text-Funktionen

4.3.5.17 OemToAnsi Prozedur

Syntax

```
OemToAnsi(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt die Zeichenkette vom OEM- (DOS-) Zeichensatz in den ANSI- (Windows-) Zeichensatz um und liefert dies als Ergebnis. Diese Funktion wird benötigt, wenn Sie Text aus einer DOS-Datei in eine TurboDB-Tabelle schreiben wollen.

Laufzeitfehler

Keine

Beispiel

Die erste Zeile einer OEM-Textdatei lesen und nach ANSI konvertieren:

```
vardef h: Integer;
vardef s: String;
h := Reset("TDATEI.TXT");
s := ReadLn(h);
s := OemToAnsi(s);
Close(h);
..Jetzt kann man s in eine Tabelle schreiben
```

Siehe auch

AnsiToOem, Text-Funktionen

4.3.5.18 Pos Prozedur

Syntax

```
Pos(SuchZeichenkette, Zeichenkette: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert die Nummer desjenigen Zeichens, ab dem *Zeichenkette* erstmals mit *SuchZeichenkette* übereinstimmt. Ist keine Übereinstimmung zu finden, liefert die Funktion den Wert 0.

Laufzeitfehler

Keine

Beispiel

```
Pos("ei","Meier") -> 2
Pos("Mei","Meier")-> 1
Pos("ay","Meier") -> 0
```

Siehe auch

Text-Funktionen

4.3.5.19 RightStr Prozedur

Syntax

```
RightStr(Zeichenkette: String, Zahl: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Liefert den rechten Teil einer *Zeichenkette*. Die Anzahl der Zeichen wird durch *Zahl* bestimmt. Sollen mehr Zeichen als vorhanden abgetrennt werden, wird der Ausgangsstring zurückgeliefert.

Laufzeitfehler

Keine

Beispiel

```
RightStr("Hans Huber",5)   ->  "Huber"
RightStr("Hans Huber",100) ->  "Hans Huber"
RightStr("Hans Huber,1)    ->  "r"
```

Siehe auch

LeftStr, Text-Funktionen

4.3.5.20 RTrim Prozedur

Syntax

```
RTrim(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Liefert die Zeichenkette wieder zurück, nachdem alle rechten Leerzeichen entfernt wurden.

Laufzeitfehler

Keine

Beispiel

```
RTrim("Hans Huber    ")  ->  "Hans Huber"
```

Siehe auch

LTrim, Text-Funktionen

4.3.5.21 Scan Prozedur

Syntax

```
Scan(Zeichenkette1,Zeichenkette2: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Zählt, wie oft eine Zeichenkette in einer anderen vorkommt. In der DOS-Version heißt diese Funktion *Zeichen*.

Laufzeitfehler

Keine

Beispiel

```
Scan("e","Hintermeier")  -> 3
```

Siehe auch

Text-Funktionen

4.3.5.22 SoundEx Prozedur

Syntax

```
SoundEx(s: String): String
```

Kategorie

Basisfunktion

Erklärung

Mit dieser Funktion wird die "phonetische Suche" realisiert. Zwei phonetisch ähnliche Begriffe sollten den gleichen Code ergeben. Die Bedingung

```
SOUNDEX(S1) = SOUNDEX(S2)
```

müßte demnach für phonetisch gleiche Begriffe zutreffen. Wenn man sich den einfachen Algorithmus der *SoundEx*-Funktion anschaut, wundert man sich nicht mehr über die manchmal recht abenteuerlichen Ergebnisse:

Die *SoundEx*-Funktion liefert einen bis zu vierstelligen Stringcode. Die erste Stelle des Codes ist der Anfangsbuchstabe des Strings S. Die weiteren Zeichen werden nach folgender Vorschrift codiert:

b, f, p, v	1
c, g, k, q, s, x, z, ß	2
d, t	3
l	4
m, n	5
r	6

Alle anderen Buchstaben oder Sonderzeichen geben keinen Code. Nachbarzeichen, die gleichen Code erzeugen, werden nur einmal codiert.

Im Gegensatz zu der in anderen Datenbanken implementierten Original *SoundEx*-Funktion wird hier das "j" nicht als 2, sondern überhaupt nicht codiert, was im Deutschen leicht verbesserte Ergebnisse zur Folge hat.

Laufzeitfehler

Keine

Siehe auch

Text-Funktionen

4.3.5.23 Str Prozedur

Syntax

```
Str(Wert: Real/Integer; Breite, Nachkommastellen: Integer; TausenderTrenner,
Füllzeichen, Dezimaltrenner: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt den Wert in Zeichenkette um. Mit Ausnahme des Wertes sind alle weiteren Parameter optional. Die *Breite* gibt die Länge der Zeichenkette an, innerhalb derer der Wert rechtsbündig dargestellt wird. Reicht die Länge zur Darstellung der Zahl nicht aus, wird die Zeichenkette automatisch soweit nötig erweitert. *Nachkommastellen* bestimmt die Anzahl der Nachkommastellen. *TausenderTrenner* gibt die Tausendertrennung an, *Füllzeichen* ist dasjenige Zeichen, mit dem die Zeichenkette nach links hin aufgefüllt wird. Wenn ein Parameter nicht angegeben ist, werden die folgenden Vorgabewerte benutzt:

Breite	1	so breit wie nötig
Nachkommastellen	0	keine Nachkommastellen
TausenderTrenner	leer	keine Tausendertrennung
Füllzeichen	Leerzeichen	vorne Leerzeichen
Dezimaltrenner	leer	das mit SetNumberFormats eingestellte Zeichen, Standard-mäßig ein Punkt

Wenn das Argument bei Wert ein Aufzählungsfeld ist, liefert *Str* den Aufzählungswert.

Wenn das Argument ein Null-Wert ist, liefert Str einen Leerstring.

Laufzeitfehler

Keine

Beispiel

```
Str(1)                  ->          "1"
Str(123)                ->        "123"
Str(1,5)                ->      "    1"
Str(123,5)              ->      "  123"
Str(1,5,2)              ->      " 1.00"
Str(123,5,2)            ->     "123.00"
Str(1,5,2,".")          ->      " 1,00"
Str(12345,12,2,".")           -> "   12.345,00"
Str(12345,12,2,".","$")       -> "$$$12.345,00"
Str(12345,12,2,".","*")       -> "***12.345,00"
```

Siehe auch

Val, Text-Funktionen, SetNumberFormats

4.3.5.24 StrAdd Prozedur

Syntax

```
StrAdd(Zeichenkette1, Zeichenkette2: String): String
```

Kategorie

Basisfunktion

Erklärung

Bei beiden Zeichenketten muss es sich um die Darstellung von ganzen Zahlen handeln. Die beiden Zahlen werden addiert und das Ergebnis wieder als String geliefert. Mit *StrAdd* ist eine Addition bis 40 Stellen möglich.

Laufzeitfehler

Keine

Beispiel

```
StrAdd("1","2")                                                -> "3"
```

```
$(StrAdd("12345678901234", "12345678901234"))             -> "24691357802468"
```

Siehe auch

Text-Funktionen

4.3.5.25 StrComp Prozedur

Syntax

```
StrComp(Zeichenkette1,Zeichenkette2: String): Real
```

Kategorie

Basisfunktion

Erklärung

Liefert einen relativen Vergleich zweier Zeichenketten. Das Ergebnis ist 0, wenn die beiden Zeichenketten in nicht einmal einem Zeichen übereinstimmen, und 1, wenn sie absolut identisch sind. In allen anderen Fällen ergibt sich ein Wert dazwischen, der um so höher ist, je mehr Fragmente der einen Zeichenkette in der anderen zu finden sind. Die genaue Formel lautet:

```
StrComp(S1,S2)  =  2*n/(Length(S1)+Length(S2))
```

wobei n die Anzahl der gleiche Zeichen (gerichtet von links nach rechts) ist.

```
String1         String2          gleiche Zeichen  StrComp
---------------------------------------------------------
Hinterhuber     Einmeier               5             0.53
 ^^ ^    ^^      ^^ ^ ^^
Sindelfingen    Sindlefingen          11             0.92
^^^^ ^^^^^^^    ^^^^^ ^^^^^^
Huberstraße     Albertusstr            6             0.55
  ^^^^^^          ^^^   ^^^
```

Die Funktion eignet sich hervorragend dazu, den zu einer Zeichenkette ähnlichsten aus einer Tabelle zu suchen.

Laufzeitfehler

Keine

Beispiel

Die Tabelle KUNDEN wird über den Ortsnamen mit der Tabelle PLZNEU gekoppelt. In einem Subreport wird der ähnlichste Straßennamen gesucht und, falls der Vergleichswert wenigstens 0.8 ergibt, die neue Postleitzahl in das Feld KUNDEN.PLZ übernommen.

```
.REPORT
.PROLOGUE
.PRIMTABLEIS KUNDEN
.RELATION KUNDEN[Ort]=PLZNEU[Ort]
.SELECTION KUNDEN.Land in ["D","W","O"], Length(KUNDEN.PLZ)<5
..Nur die bislang unbehandelten Postleitzahlen werden bearbeitet
.DATA
.VAR MaxStrComp=0
.VAR PLZhierzu=""
.SUB PLZNEU
.  VAR vgl=StrComp(KUNDEN.Straße,PLZNEU.Straße)
.  IF vgl>MaxStrComp
.    VAR MaxStrComp=vgl
.    VAR PLZhierzu=PLZNEU.PLZ
.  END
.ENDSUB
.IF MaxStrComp>=0.8
$(KUNDEN.Ort:25 KUNDEN.Straße:25 KUNDEN.PLZ:5 " -> " PLZhierzu)
.REPLACE KUNDEN(PLZ=PLZhierzu)
.EPILOGUE
```

Siehe auch

Text-Funktionen

4.3.5.26 RealVal Prozedur

Syntax

```
procedure RealVal(S: String): Real;
```

Kategorie

Basisfunktion

Erklärung

Konvertiert den String in eine Zahl. Falls der String keine Zahl enthält, wird ein Laufzeitfehler ausgelöst. Im Unterschied zur Funktion *Val* werden nur Zahlen und nicht beliebige nummerische Ausdrücke ausgewertet.

Laufzeitfehler

44 S ist ein Datum oder eine Uhrzeit.

46 S ist keine gültige Zahl.

Beispiele

```
RealVal('2.8')          -> 2,8
RealVal('57')     -> 57,0
RealVal('3 * 5') -> Laufzeitfehler
RealVal('13:45') -> Laufzeitfehler
```

Siehe auch

Val

4.3.5.27 Subst Prozedur

Syntax

```
procedure Subst(S: String, T: String [N: Integer]): Integer
procedure Subst(S: String, Tabelle: Integer, FeldNummer: Integer [, Modus:
Integer]): Integer
```

Kategorie

Basisfunktion

Erklärung

Ersetzt *S* im Ramtext durch *T* bzw. durch den Wert des Feldes *FeldNummer* in der Tabelle. Falls der Modus 1 ist, wird der Feldwert dabei noch von HTML in normalen Text konvertiert.

Laufzeitfehler

Keine

Siehe auch

Exchange

4.3.5.28 Swap Prozedur

Syntax

```
Swap(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Vertauscht die *Zeichenkette* bzgl. des "^"-Zeichens. Diese Funktion ist wichtig im Zusammenhang mit dem Ordnungssteuerzeichen "^", das in Stringfeldern zur Definition des Sortierstartpunkts verwendet wird. Ein Stringfeld des Inhalts "Hubert^Müller GmbH" wird im Index wie "Müller GmbH^Hubert" eingeordnet. Stringvergleich mittels Relationsoperatoren wie "kleiner" oder "ab" berücksichtigen automatisch den Startpunkt. In eigenen Anwendungen kann es freilich nötig werden, auf die interne Ordnung zurückzugreifen.

Laufzeitfehler

Keine

Beispiel

```
Swap("A^B")   ->  "B^A"
Swap("ABC")   ->  "ABC"
```

Wenn man die Funktion *Swap* zweimal auf eine beliebige Zeichenkette anwendet, kommt wieder die ursprüngliche Zeichenkette heraus:

```
Swap(Swap(Zeichenkette))  -> Zeichenkette
```

Siehe auch

Text-Funktionen

4.3.5.29 TestLn Prozedur

Syntax

```
TestLn(s: String): String
```

Kategorie

Basisfunktion

Erlärung

An die übergebene Zeichenkette wird ein Zeilenumbruch und ein Zeilenvorschub angehängt (Chr(13) und Chr(10) unter Windows). Wird eine leere Zeichenkette übergeben, passiert nichts.

Laufzeitfehler

Keine

Beispiel

```
Write(Hdl, TestLn("Irgendein Text"))
```

hat denselben Effekt wie

```
WriteLn(Hdl, "Irgendein Text")
```

Siehe auch

Text-Funktionen

4.3.5.30 ToHtml Prozedur

Syntax

```
ToHtml(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt die Zeichenkette vom ANSI- (Windows-) Zeichensatz in das HTML-Format und liefert dies als Ergebnis. Die Funktion wird benötigt, wenn Sie Text aus einer Windows-Datei einlesen und für eine WEB-Site verwenden wollen..

Laufzeitfehler

Keine

Beispiel

```
procedure NachHTMLWandeln
      vardef S: String;
      S := "Willkommen bei den Würdinger Möbelwerken"
      Message(ToHtml(S))
endproc
```

Der Text wird somit folgendermaßen ausgegeben:

```
Willkommen bei den W&uuml;rdinger M&ouml;belwerken
```

Siehe auch

Memo2HTML, Text-Funktionen

4.3.5.31 Upper Prozedur

Syntax

```
Upper(Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt die Zeichenkette in Großschreibung. Alle Zeichen der Zeichenkette außerhalb des Buchstabenbereichs werden nicht umgewandelt. Das Zeichen "ß" wird ebenfalls nicht behandelt. Soll dieses Zeichen in "SS" umgewandelt werden, so kann das durch eine Kombination dieser Funktion mit *Exchange* erfolgen.

Diese Funktion ist damit das genaue Gegenstück zur Funktion *Lower*

Laufzeitfehler

Keine

Beispiel

```
Upper("Das ist eine Überschrift")          -> "DAS IST EINE ÜBERSCHRIFT"
Upper("Titel in Großbuchstaben")           -> "TITEL IN GROßBUCHSTABEN"
Exchange(Upper("Großbuchstaben"),"ß","SS") -> "GROSSBUCHSTABEN"
```

Siehe auch

Lower, Text-Funktionen

4.3.5.32 Val Prozedur

Syntax

```
Val(Zeichenkette: String): Real
```

Kategorie

Basisfunktion

Erklärung

Ermittelt den Wert einer Zeichenkette. Die Zeichenkette wird als Ausdruck interpretiert und berechnet. Der Wert dieses Ausdrucks bildet das Ergebnis der Funktion. Falls der Ausdruck nicht zu einer Zahl ausgewertet werden kann, wird ein Laufzeitfehler ausgelöst.

Achtung

Diese Funktion ist sehr mächtig, weil sie beliebige Ausdrücke ausrechnen kann. Dadurch ist sie aber auch sehr gefährlich. Wenn die Zeichenkette vom Benutzer eingegeben werden kann, kann dieser praktisch alle Operationen auf der Datenbank ausführen. Für die meisten Fälle, wo eine Zeichenkette in eine Zahl umgewandelt werden soll, sollte man *RealVal* einsetzen.

Laufzeitfehler

Diverse	Die Zeichenkette enthält keinen berechenbaren Ausdruck.

Beispiel

```
Val("12")            -> 12
Val("12" + "34")     -> 1234
Val("12 + 34")       -> 46
DateStr(Val("1.1.2000"))      -> "01.01.2000"
```

Siehe auch

RealVal, DateTimeVal, ValStr, Text-Funktionen

4.3.5.33 ValStr Prozedur

Syntax

```
ValStr(Ausdruck: String): String
```

Kategorie

Basisfunktion

Erklärung

ValStr wertet den übergebenen Ausdruck aus und gibt, falls möglich, das Ergebnis als Zeichenkette zurück.

Laufzeitfehler

Diverse	Die Zeichenkette enthält keinen berechenbaren Ausdruck.

Beispiel

```
T-Eingabe := "KUNDEN.Name";
Message(ValStr(T-Eingabe));
..Zeigt den String "Bolte" an
```

Siehe auch

Val, Text-Funktionen

4.3.6 Dateien und Ordner

4.3.6.1 Dateien und Ordner

Dateien und Verzeichnis abfragen, anlegen, beschreiben, auslesen, schließen und vieles mehr.

4.3.6.2 BaseDir

Syntax

```
BaseDir: String
```

Kategorie

Basisfunktion

Erklärung

Liefert das Verzeichnis in dem sich die aktuelle Projektdatei befindet

Beispiel

```
Message("Das Projektverzeichnis ist: " + BaseDir);
```

Siehe auch

PrivDir, DBDir

4.3.6.3 ChDir

Syntax

```
ChDir(Verzeichnis: String): Integer
```

Kategorie

Basisfunktion

Erklärung

ChDir wechselt in das angegebene Verzeichnis. Der Rückgabewert ist 0 wenn der Wechsel erfolgreich war, andernfalls 2.

Beispiel

```
ChDir('C:\TurboDB Studio\BIN')
```

Siehe auch

GetDir, MakeDir, RemDir

4.3.6.4 Close

Syntax

```
Close(FileNo: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

Die Zahl entspricht einem Texthandle, der mit einer der Funktionen Reset, Rewrite oder TAppend erzeugt wurde. *Close* schließt die zugehörige Textdatei und liefert 0 im Erfolgsfall beziehungsweise einen positiven Fehlercode.

Beispiel

```
vardef t: Integer;
t := Reset(T-Eingabe);
...Hier wird aus der Datei gelesen
Close(t);
```

Siehe auch

Eot, Read, ReadLn, Reset, Rewrite, TAppend, Write, WriteLn

4.3.6.5 CloseFindFile

Syntax

```
CloseFindFile
```

Kategorie

Basisfunktion

Beschreibung

Die Funktion gibt den zuvor mit *FindFirstFile* reservierten Speicher frei. Sie muss nach jeder Verwendung von *FindFirstFile* aufgerufen werden, weil der Speicherverbrauch der Anwendung sonst ständig zunimmt.

Beispiel

siehe FindFirstFile

Siehe auch

FindFirstFile, FindNextFile

4.3.6.6 CopyFile

Syntax

```
CopyFile(Source, Destination: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Die Datei *Source* wird kopiert. Die neue Datei wird durch Destination bezeichnet. Im Erfolgsfall wird 0 zurückgegeben, sonst der (positive) Windows-Fehlercode.

Beispiel

```
if CopyFile('C:\VDP\README.TXT', 'A:\LESEN.TXT') = 0
   Message('Die Datei wurde erfolgreich kopiert')
else
   Message('Beim Kopieren der Datei ist ein Fehler aufgetreten')
end
```

Siehe auch

DelFile, Reset, Rewrite, Close

4.3.6.7 DelFile

Syntax

```
DelFile(Zeichenkette: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Die Datei mit dem angegeben Namen wird gelöscht. Das Ergebnis der Funktion ist 0, wenn die

Operation erfolgreich ausgeführt werden konnte, andernfalls wird der Fehlerstsatus zurückgeliefert.

Beispiel

Das Makro "Anschreiben" exportiert Informationen aus der Tabelle KUNDEN in dein Textdatei, startet dann ein externes Programm, das mit dieser Textdatei arbeitet und löscht schließlich die Textdatei wieder.

```
PROCEDURE Anschreiben
      VARDEF T, i : REAL
      VARDEF TempName : STRING
      TempName:="TEMP$$$.TXT"
      IF T:=Rewrite(TempName)>0
           Link(KUNDEN,WriteLn(T,Name+","+Vorname))
           Close(T)
           Execute("D:\PROGS\MERGE.EXE",TempName)
           DelFile(TempName)
      END
ENDPROC
```

Siehe auch

Reset, Rewrite, Close, CopyFile, IsFile

4.3.6.8 DiskFree

Syntax

```
DiskFree(Drive: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

DiskFree liefert den freien Platz auf der Diskette/Festplatte in Bytes. Die Zahl bestimmt das Laufwerk:

0 = aktuelles Laufwerk
1 = A:
2 = B:
3 = C:
...

Das Ergebnis -1 erhält man, falls das angegebene Laufwerk nicht vorhanden bzw. keine Diskette eingelegt ist.

Beispiel

```
WHILE DiskFree(3) < 1000000
      Message("Diese Anwendung benötigt mindestens 1 MB auf der Festplatte.",
"Achtung", 1)
END
```

Siehe auch

GetDrive

4.3.6.9 Eot

Syntax

```
Eot(FileHdl: Integer): Real
```

Kategorie

Basisfunktion

Erklärung

Bei der Zahl handelt es sich um einen Texthandle, der mit der Funktion Reset ermittelt wurde. Eot liefert den Wert 1, falls das Dateiende erreicht wurde, andernfalls 0.

Beispiel

Im einem Datenbankjob wird eine externe Textdatei (EXTERN.TXT) ausgedruckt. (TurboPL-Befehle im Datenbankjob werden mit Punkt gekennzeichnet.)

```
.PROLOGUE
.VARDEF Text: REAL
.IF Text := Reset("EXTERN.TXT")
.     WHILE NOT Eot(Text)
      $(ReadLn(Text))
.     END
.     DO Close(Text)
.END
```

Siehe auch

Close, Read, ReadLn, Reset, Rewrite, TAppend, Write, WriteLn

4.3.6.10 FindFirstFile

Syntax

```
FindFirstFile(Mask, AttrMask: String; var FileName: String; var FileSize:
Integer; var FileDate: Date; var FileTime: Time; var Attr, Folder: String):
Integer;
```

Kategorie

Basisfunktion

Beschreibung

Sucht den ersten Verzeichniseintrag, der dem in *Mask* übergebenen Suchmuster und den Attributen in *AttrMask* übereinstimmt.

Die Attribute in *AttrMask* bestimmen, welche Einträge zusätzlich zu den normalen Dateien gesucht werden sollen. Es ist eine Kombination aus den Datei-Attribut-Buchstaben, die weiter unten beschrieben sind. Ein Leerstring '' sucht also nur nach normalen Dateien, 'D' sucht nach normalen Dateien und Verzeichnissen. 'SH' sucht nach normalen Dateien, Systemdateien und versteckten Dateien usw.

Das Ergebnis der Suche wird in den Parametern *FileName* bis *Folder* zurückgegeben:

FileName	Name der gefundenen Datei
Size	Dateigrösse
Date	Datumstempel der Datei
Time	Zeitstempel der Datei
Attr	Dateiattribute als Zeichenkette von Datei-Attribut-Buchstaben
Folder	Ordner, in dem sich die Datei befindet

Ist die Suche erfolgreich, wird 0 zurückgegeben, ansonsten ein negativer Fehlercode:

-1	Allgemeiner Fehler
-2	Der Pfad in Mask ist ungültig, zum Beispiel weil er ein nicht vorhandenes Laufwerk oder Verzeichnis enthält.
-3	Es wurde kein passender Verzeichniseintrag gefunden.

Um den nächsten Verzeichniseintrage zu lesen, benutzen Sie die Funktion *FindNextFile*. Wenn Sie auch den Inhalt von Unterverzeichnissen benötigen, müssen Sie *FindFirstFile* rekursiv aufrufen, wie im Beispiel unten gezeigt wird.

Achtung

FindFirstFile belegt Ressourcen, die durch einen Aufruf von *CloseFindFile* wieder freigegeben werden müssen.

Datei-Attribute

Die folgenden Buchstaben kennzeichnen die verschiedenen Datei-Attribute. Kombinationen dieser Buchstaben werden sowohl bei der Angabe von *AttrMask* als auch im Ergebnis als Attr verwendet. Dabei kommt es auf die Reihenfolge der Buchstaben im String nicht an:

D	Verzeichnis (Directory)
H	Versteckt (Hidden)
S	System-Datei
V	Datenträger-Bezeichnung (Volume ID, nur unter DOS wirksam)

R Nur lesen (Read Only), hat als AttrMask keine Wirkung

A Archiv, hat als AttrMask keine Wirkung

Beispiel

Der folgende Code schreibt den Inhalt des übergebenen Ordners inklusiv aller Unterverzeichnisse in die Datei content.txt. Dazu ruft man *ListFolderContent* mit dem gewünschten Ursprungsverzeichnis auf, z.B. *ListFolderContent("C:\programme")*:

```
procedure WriteFolderContent(OutFile: Integer; Level: Integer; Folder:
String);
    vardef Result: Integer;
    vardef FName, FAttributes, FFolder: string;
    vardef FSize: Integer;
    vardef FDate: Date;
    vardef FTime: Time;
    Result := FindFirstFile(Folder + "\*.*", "DHS", FName, FSize, FDate, FTime,
FAttributes, FFolder);
    while Result = 0
       if not FName = "." and not FName = ".."
          WriteLn(OutFile, NTimes("  ", Level) + FName + " " + DateStr(FDate) +
" " + FAttributes);
          if Pos("D", FAttributes) > 0
             WriteFolderContent(OutFile, Level + 1, Folder + "\" + FName);
          end;
       end;
       Result := FindNextFile(FName, FSize, FDate, FTime, FAttributes,
FFolder);
    end;
    CloseFindFile;
endproc;

procedure ListFolderContent(StartFolder: String);
    vardef OutHdl: Integer;
    OutHdl := ReWrite(StartFolder + "\content.txt");
    WriteLn(OutHdl, "Content of Folder: " + StartFolder);
    WriteFolderContent(OutHdl, 0, StartFolder);
    Close(OutHdl);
endproc;
```

Siehe auch

FindNextFile, CloseFindFile

4.3.6.11 FindNextFile

Syntax

```
FindNextFile(var FileName: String; var FileSize: Integer; var FileDate: Date;
var FileTime: Time; var Attr, Folder: String): Integer;
```

Kategorie

Basisfunktion

Beschreibung

FindNextFile liefert den nächsten Eintrag, der mit dem Dateinamen und den Attributen übereinstimmt, die zuvor im Aufruf von *FindFirstFile* angegeben wurden. Die Rückgabewerte entsprechen denen von *FindFirstFile* (d.h. 0 bei Erfolg, sonst negativer Fehlercode).

Wenn *FindNextFile* ohne ein vorangehendes *FindFirstFile* aufgerufen wird, wir ein Laufzeitfehler ausgelöst.

Beispiel

siehe FindFirstFile

Siehe auch

FindFirstFile, CloseFindFile

4.3.6.12 FirstDir

Syntax

```
FirstDir(Suchmuster: String; Attribute: String): String
```

Diese Funktion existiert lediglich aus Kompatibilitätsgründen und kann keine langen Dateinamen verarbeiten. Bitte verwenden Sie zukünftig nur die erweiterten Funktionen FindFirstFile und FindNextFile.

Kategorie

Basisfunktion

Erklärung

Sucht den ersten Verzeichniseintrag, der dem Suchmuster und den Attributen entspricht und liefert einen String mit dem Dateinamen, der Dateigröße und weiteren Angaben zurück. Das Suchmuster enthält normalerweise Joker-Zeichen wie z.B. "*.*" oder "*.DAT" oder "MAI???.K*". Attribute gibt an, welche Dateien zusätzlich zu den normalen Dateien gesucht werden sollen und stimmt mit den Attributen im Datei-Manager überein:

R	Nur lesen (Read Only)
A	Archiv
S	System-Datei
H	Versteckt (Hidden)
V	Datenträger-Bezeichnung (Volume ID)
D	Verzeichnisse (Directory)

Anstelle eines Attribut-Strings kann auch das Zeichen "X" angegeben werden. In diesem Fall enthält der Rückgabestring den kompletten Pfad zur angegebenen Datei.

Der Rückgabe-String enthält Angabe über die gefundene Datei:

Zeichen	Inhalt
1..12	Dateiname mit Extension
15..24	Dateigröße in Byte
27..36	Datei-Datum
39..43	Datei-Zeit
46	"R" bei Nur-lesen-Dateien, ansonsten leer
47	"H" bei versteckten Dateien, ansonsten leer
47	"S" bei System-Dateien, ansonsten leer
47	"V" bei Datenträger-Bezeichnung, ansonsten leer
47	"D" bei Verzeichnissen, ansonsten leer
47	"A" bei Archiv-Dateien, ansonsten leer
54..	Verzeichnis-Name

Beispiel

Ausdruck des aktuellen Verzeichnisses:

```
procedure VerzeichnisDrucken;
    vardef S: String;
    S := FirstDir("*.*", "");
    while S <> ""
       Print(S/);
       S := NextDir;
    end
endproc
```

Siehe auch

NextDir

4.3.6.13 GetDir

Syntax

```
GetDir(Zahl: Real): String
```

Kategorie

Basisfunktion

Erklärung

Liefert den Namen des aktuellen Verzeichnisses auf dem Laufwerk mit der angegebenen Laufwerksnummer. Dabei ist:

0: Aktuelles Laufwerk

1: Laufwerk A:

2: Laufwerk B:

3: Laufwerk C: usw.

Beispiel

```
Message(GetDir(17))
```

Die Ausgabe könnte z.B. Q:\WEBREP\KFZ lauten. (Q ist der 17. Buchstabe im Alphabet.)

Siehe auch

ChDir, GetDrive, MakeDir, RemDir

4.3.6.14 GetDrive

Syntax

```
GetDrive(Laufwerk: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert den Status eines Laufwerks. Dabei gilt 0=A:, 1=B: usw.

Der Rückgabewert ist:

0 : Laufwerkstyp nicht bestimmbar

1 : Laufwerk nicht vorhanden/kein Root-Verzeichnis

2 : Wechselmedium/Diskettenlaufwerk

3 : fest eingebautes Medium/lokale Festplatte

4 : logisches Laufwerk/Netzwerklaufwerk

5 : CD-Laufwerk

6 : Ramdisk

Beispiel

```
if GetDrive(1) <= 1
   Message("Laufwerk B: nicht vorhanden.");
end
```

Zur Ermittlung der Laufwerksnummer läßt sich folgende Prozedur verwenden:

```
procedure GetDriveNo(LaufwerksBuchstabe: String): Integer
   vardef LWNummer: Integer;
   LWNummer := Asc(Upper(LaufwerksBuchstabe)) - Asc("A") + 1
   if (LWNummer < 0) or (LWNummer > 25)
      return -1 ..ungültig
   else
      return LWNummer
   end
endproc
```

Siehe auch

GetDir, Asc, Upper

4.3.6.15 GetSize

Syntax

```
GetSize(Datei: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Die Größe der Datei wird zurückgegeben (in Anzahl Bytes). Im Fehlerfall ist der Rückgabewert -1.

Beispiel

```
vardef Size: Real;
Size := GetSize("C:\VDP\VDP.EXE");
Message("VDP.EXE ist " + Str(Size) + " Bytes groß");
```

Siehe auch

CopyFile, DelFile, IsFile

4.3.6.16 IsFile

Syntax

```
IsFile(Zeichenkette: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Die Zeichenkette bestimmt einen Dateinamen. Die Funktion liefert 1, falls eine Datei dieses Namens existiert, andernfalls 0. Die Funktion heißt in der DOS-Version *Exists*.

Beispiel

Im folgenden Makro wird geprüft, ob eine Datei bereits existiert. Im positiven Fall kann der Anwender bestimmen, ob die Datei überschrieben wird oder nicht.

```
T-Eingabe:=""
repeat
   Input("Name der Ausgabedatei","Export")
until not IsFile(T-Eingabe) or Message("Datei existiert bereits!
Überschreiben?","Achtung",3)=6
```

Siehe auch

Reset, Rewrite, Close, DelFile, CopyFile, GetSize

4.3.6.17 MakeDir

Syntax

```
MakeDir(Zeichenkette: String): Real
```

Kategorie

Basisfunktion

Beschreibung

Die Zeichenkette muss ein dem Betriebssystem entsprechender Verzeichnisname sein. Es wird ein Verzeichnis mit diesem Namen neu angelegt.

Ergebnis

0	Verzeichnis erfolgreich angelegt
1	Verzeichnis existiert bereits
2	Verzeichnis kann nicht angelegt werden

Beispiel

```
IF MakeDir("D:\TEXTE")=1
      Message("Verzeichnis existiert bereits","Achtung",1)
END
```

Siehe auch

ChDir, GetDir, RemDir

4.3.6.18 NextDir

Syntax

```
NextDir: String;
```

Kategorie

Basisfunktion

Erklärung

Kann nur nach einem vorangegangenen FirstDir aufgerufen werden. Liefert die Beschreibung des nächsten passenden Verzeichniseintrags. Diese Funktion existiert lediglich aus Kompatibilitätsgrunden und kann keine langen Dateinamen verarbeiten. Bitte verwenden Sie zukünftig nur die erweiterten Funktionen FindFirstFile und FindNextFile.

Beispiel

siehe FirstDir.

4.3.6.19 NHandles

Syntax

```
NHandles: Integer
```

Kategorie

Basisfunktion

Erklärung

TurboDB Studio kann nur eine begrenzte Anzahl von Textdateien mittels *Reset/Rewrite* gleichzeitig öffnen. *NHandles* gibt zurück, wieviele Dateien noch geöffnet werden können.

Beispiel

```
Message(Str(NHandles))   -> 50
Rewrite("Hallo.txt")
Message(Str(NHandles))   -> 49
```

Siehe auch

Reset, Rewrite, Close

4.3.6.20 OrdnerAuswählen

Syntax

```
OrdnerAuswählen(Überschrift: String [; var OrdnerName: String]): Integer
ChooseFolder(Title: String; [; var FolderName: String]): Integer
```

Kategorie

Basisfunktion

Beschreibung

Zeigt ein Dialogfenster für die Auswahl eines Ordners mit der Überschrift Überschrift an. *OrdnerName* liefert beim Aufruf die Vorbelegung und gibt nach Beendigung der Funktion den gewählten Ordner zurück. Wenn *OrdnerName* nicht angegeben ist, wird die Systemvariable *T-Eingabe* benutzt.

Der Rückgabewert beträgt 1, falls die Eingabe bestätigt wurde.

Beispiel

```
procedure VerzeichnisAussuchenUndAnzeigen
    vardef Result: Integer;
    vardef FolderName: String;
    ..Vorbelegung
    FolderName := "C:\VDP\";
    ..Dialog ausführen
    Result := OrdnerAuswählen("Das wichtigste Verzeichnis", FolderName)
    ..Auf Eingabe reagieren
```

```
    if Result = 1
       Message("Das wichtigste Verzeichnis ist: " + FolderName);
    else
       Message("Abbruch gedrückt");
    end
endproc
```

Siehe auch

DateiAuswählen, Input

4.3.6.21 PrivDir

Syntax

```
PrivDir: String
```

Kategorie

Basisfunktion

Erklärung

Liefert das private Verzeichnis, das in der Netzwerkversion über den Menüpunkt *Datei/Einstellungen* vergeben wird.

Beispiel

```
Message("Das private Verzeichnis ist: " + PrivDir);
```

Siehe auch

BaseDir, DBDir, NetUsers, UserNo

4.3.6.22 Read

Syntax

```
Read(FileHandle, Count: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Bei File*Handle* handelt es sich um einen über die Funktion *Rese*t ermittelten Datei-Handle. Das Ergebnis der Funktion ist die gelesene Zeichenkette. Wird über das Ende der Datei hinaus gelesen, so werden entsprechend weniger Zeichen gelesen und die Funktion *Eot* liefert den Wert 1.

Count gibt an, wieviele Zeichen gelesen werden. Ist er nicht vorhanden, so wird nur ein Zeichen gelesen. Der Wert -1 bedeutet, dass soviele Zeichen gelesen werden, vorhanden sind.

Laufzeitfehler

2	Datei ist defekt oder gesperrt.
37	Der Datei-Handle ist ungültig.

Anmerkung

In den Versionen bis TurboDB Studio 3 waren hier höchstens 255 Zeichen pro Aufruf zugelassen. Diese Beschränkung gilt nicht mehr.

Beispiel

Schnelle Kopie einer Textdatei, wobei die Zeilen auch länger als 255 Zeichen sein können.

```
VAR Dateiname = "EXTERN.TXT"
VARDEF Eingabe, Ausgabe : REAL
IF Eingabe := Reset(Dateiname) > 0
      IF Ausgabe := Rewrite("A:\"+Dateiname)
           WHILE NOT Eot(Eingabe)
                Write(Ausgabe, Read(Eingabe,255))
           END
           Close(Ausgabe)
      ELSE
           Message("A:\"+Dateiname+" kann nicht erzeugt werden","Fehler",1)
      END
      Close(Eingabe)
```

```
ELSE
      Message(Dateiname+" nicht gefunden","Fehler",1)
END
```

Siehe auch

Close, Eot, ReadLn, Reset, Rewrite, TAppend, Write, WriteLn

4.3.6.23 ReadLn

Syntax

```
ReadLn(FileHandle: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Aus der zugehörigen Textdatei wird eine komplette Zeile bis zum nächsten CR/LF gelesen. Diese Zeile bildet das Ergebnis der Funktion. Der Zeilenvorschub (=CR/LF) wird überlesen.

FileHandle ist ein von *Reset* zurückgeliefertes Datei-Handle.

Laufzeitfehler

2	Datei ist defekt oder gesperrt.
37	Der Datei-Handle ist ungültig.

Anmerkung

In früheren Versionen wurden maximal 255 Zeichen der eingelesenen Zeile als Funktionswert geliefert. Diese Beschränkung gibt es nicht mehr.

Beispiel

Die Prozedur kopiert eine Textdatei:

```
procedure Kopiere_Text(Quelldatei, Zieldatei: String)
vardef QuellHdl, ZielHdl: Real;
      QuellHdl := Reset(Quelldatei);
      ZielHdl := Rewrite(Zieldatei);
      IF QuellHdl > 0 and ZielHdl > 0
           WHILE NOT EOT(QuellHdl)
                WriteLn(ZielHdl, ReadLn(QuellHdl));
           END;
      END;
      Close(QuellHdl);
      Close(ZielHdl);
endproc
```

Siehe auch

Close, Eot, Read, Reset, Rewrite, TAppend, Write, WriteLn

4.3.6.24 RemDir

Syntax

```
RemDir(Zeichenkette: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Löscht ein Verzeichnis. Bei der Zeichenkette muss es sich um einen gültigen Verzeichnisnamen handeln. Dieses Verzeichnis wird gelöscht.

Ergebnis:

Verzeichnis wurde entfernt	-> 0
Verzeichnis nicht leer	-> 1
Verzeichnis existiert nicht	-> 2

Beispiel

```
RemDir("C:\DATEN\TEMP")
```

Siehe auch

ChDir, GetDir, MakeDir

4.3.6.25 Rename

Syntax

```
Rename(OldFileName, NewFileName: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Die durch die *OldFileName* bestimmte Datei wird in *NewFileName* umbenannt. Das Ergebnis ist 1, wenn die Umbenennung erfolgreich durchgeführt wurde, ansonsten 0.

Beispiel

```
Rename("C:\DATEN\EXTERN.TXT","C:\DATEN\INTERN.TXT")
```

Siehe auch

DelFile

4.3.6.26 Reset

Syntax

```
Reset(Dateiname: String): Integer;
```

Kategorie

Basisfunktion

Erklärung

Öffnet eine Textdatei zum Lesen. Falls die Datei erfolgreich geöffnet werden kann, liefert die Funktion einen Datei-Handle über den in der Folge auf die Datei (Read, ReadLn, Eot, Close) zugegriffen wird. Bei Misserfolg wird 0 zurückgegeben.

Wird als Dateiname RAMTEXT angegeben wird die Speicher-Dateigeöffnet. Auf diese kann genau so zugegriffen werden, wie auf eine normale Datei. Sie befindet sich jedoch ständig im Hauptspeicher, so dass die Zugriffe sehr schnell sind und auch kein Konflikt im Netzwerk auftreten kann.

Beispiel

Schreiben in ein Memo-Feld über eine Speicher-Datei

```
PROCEDURE READ_TEXT
      VarDef nFI : REAL;
      .. Speicher-Datei wird angelegt und geöffnet
      nFI     := REWRITE("RAMTEXT");
      .. Wir schreiben einen kurzen Text hinein...
      WRITELN(nFI, "Dies wird ein Text, ");
      WRITELN(nFI, "der anschließend an das Memo");
      WRITELN(nFI, "angehängt wird.");
      ...und schließen sie wieder
      CLOSE(nFI);
      .. RAMTEXT-Inhalt ans Memo anhängen
      ReadMemo(KUNDEN.Memo, "RAMTEXT", 1);
ENDPROC
```

Siehe auch

Close, Eot, IsFile, Read, ReadLn, Rewrite, TAppend, Write, WriteLn

4.3.6.27 Rewrite

Syntax

```
Rewrite(Dateiname: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Öffnet Textdatei zum Schreiben. Falls die Datei erfolgreich erzeugt werden kann, liefert die Funktion eine Zahl (=Texthandle) über die in der Folge auf die Datei (*Write*, *WriteLn*, *Close*) zugegriffen wird. Besteht bereits eine Datei mit dem angegeben Namen, so wird diese überschrieben. Soll eine Datei zum Weiterschreiben geöffnet werden, muss die Funktion *TAppend* verwendet werden. Bei Misserfolg wird 0 zurückgegeben.

Wie mit *Reset* kann auch mit *Rewrite* die Speicher-Datei geöffnet werden.

Beispiel

Siehe Read.

Siehe auch

Close, Eot, IsFile, Read, ReadLn, Reset, TAppend, Write, WriteLn

4.3.6.28 SubPath

Syntax

```
SubPath(Gesamtzeichenkette, Teilzeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Die Funktion liefert den rechten Teilstring von *Gesamtzeichenkette*, wenn man die gleichen Zeichen bzgl. *Teilzeichenkette* entfernt.

Beispiel

```
SubPath("Hans Müller", "Hans M")  ->  "üller"
```

Die Funktion wird vorzugsweise im Zusammenhang mit *LinkBlob* eingesetzt, um auf ein zur Tabelle relativen Verzeichnis zu kommen:

```
T-Eingabe := "*.BMP"
if ChooseFile("")
   LinkBlob(SubPath(T-Eingabe, DBDir(FileNo)));
end
```

Ist *T-Eingabe* beispielsweise "C:\VDP\DATEN\BILDER\TEST.BMP" und *DBDir(FileNo)* "C:\VDP\DATEN\" so wird das Bild mit dem relativen Pfad "BILDER\TEST.BMP" eingebunden und ist damit im Netz auch für anders gemappte Laufwerke verfügbar.

Siehe auch

AnsiToOem, DigitStr, Exchange, FillStr, LeftStr, Length, LTrim, OemToAnsi, Pos, Rename, RightStr, RTrim, Scan, Str, StrAdd, StrComp, StrSort, Swap, ToHtml, Upper

4.3.6.29 TAppend

Syntax

```
TAppend(Dateiname: String): Integer
```

Kategorie

Basisfunktion

Erklärung

Öffnet die Datei zum Anhängen von Daten an die bestehenden. Zurückgegeben wird im Erfolgsfall die Dateinummer, über den in der Folge auf die Datei zugegriffen werden kann. Tritt ein Fehler auf, wird 0 zurückgegeben.

TAppend sollte nicht für die Speicher-Datei aufgerufen werden. Falls dies dennoch geschieht, hat *TAppend* hier den selben Effekt wie *Rewrite*.

Beispiel

Die Prozedur hängt an das Ende der übergebenen Textdatei die Zeile "<END OF FILE>" an.

```
procedure WriteEOF(TextFile: string)
   vardef FileHdl: Integer;
   FileHdl := TAppend(TextFile)
   if FileHdl > 0
```

```
    WriteLn(FileHdl, "<END OF FILE>");
    Close(FileHdl);
    end;
endproc;
```

Siehe auch

Close, Eot, Read, ReadLn, Reset, Rewrite, Write, WriteLn

4.3.6.30 Write

Syntax

```
Write(DateiNr: Integer; Zeichenkette: String): String
```

Kategorie

Basisfunktion

Erklärung

Bei *DateiNr* handelt es sich um einen über die Funktion *Rewrite* oder *TAppend* festgelegten Datei-Nummer. Die Zeichenkette wird in die zugehörige Textdatei geschrieben. Das Ergebnis ist die Zeichenkette selbst.

Beispiel

In eine Textdatei werden alle Namen der Kunden ohne Zeilenvorschub geschrieben.

```
vardef T, erster : Integer
IF T := Rewrite("EXTERN.TXT")
    erster := 1
    sub KUNDEN
       if not erster
          Write(T,", ")
       end
       Write(T,KUNDEN.Name)
       erster:=0
    endsub
    Close(T)
end
```

Siehe auch

Close, Eot, Read, ReadLn, Reset, Rewrite, TAppend, WriteLn

4.3.6.31 WriteLn

Syntax

```
WriteLn(FileNo: Integer; Zeichenkette: String)
```

Kategorie

Basisfunktion

Erklärung

Wie *Write*, nur dass hier nach der Ausgabe der Zeichenkette noch ein Zeilenvorschub in die Textdatei geschrieben wird.

Beispiel

siehe ReadLn

Siehe auch

Close, Eot, Read, ReadLn, Reset, Rewrite, TAppend, Write, WriteLn

4.3.7 Arrays

4.3.7.1 Arrays

Funktionen zum Bearbeiten von Feldern.

4.3.7.2 ClrArray Prozedur

Syntax

```
ClrArray(var Vektor: Array)
```

Kategorie

Basisfunktion

Erklärung

Löscht alle Einträge im Array. Bei einem nummerischen Array werden die Einträge auf 0, bei einem String-Array auf Leerstring gesetzt.

Beispiel

Die Array-Einträge werden zuerst mit den geraden Zahlen vorbelegt und dann alle gelöscht.

```
vardef Vektor: Integer[100];
..Vektor vorbesetzen
vardef i: Integer;
i := 0;
while i < 100
    Vektor[i] := 2*i;
    i := i + 1;
end;
..Alle Einträge auf 0 setzen
ClrArray(Vektor);
```

Siehe auch

High, Redim

4.3.7.3 High Prozedur

Syntax

```
High(Dimension: Integer; Feld: Array): Integer
```

Kategorie

Basisfunktion

Erklärung

Bei der Variablen muss es sich um eine Arrayvariable handeln. Die Zahl gibt die Dimension an, deren höchster Feldindex zurückgegeben wird. Falls das Array leer ist, liefert die Funktion -1. Die Funktion *High* wird im Zusammenhang mit Feldparametern an Prozeduren eingesetzt.

Beispiel

```
vardef Namen: String[100]
vardef Vektor: String[50]
vardef Matrix: Real[10,20]
vardef Leer: Integer[]
High(1, Namen)   ->  100
High(1, Vektor)  ->   50
High(1, Matrix)  ->   10
High(2, Matrix)  ->   20
High(1, Leer)    ->   -1

procedure PrintStrings(var Feld: String[])
    vardef i: Integer;
    nLoop(i, High(1, Feld), Feld[i] := Str(i)))
endproc
```

Siehe auch

ClrArray, Redim

4.3.7.4 InArray Prozedur

Syntax

```
InArray(Number: Real; Array: Real[]): Integer
InArray(Number: Integer; Array: Integer[]): Integer
InArray(Str: String; Array: String[]): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert 1, wenn die Zahl bzw. die Zeichenkette im Array enthalten ist, andernfalls 0.

Beispiel

```
if InArray(1, Feld)
   Message( 'Die Zahl 1 ist im Array "Feld" enthalten');
end;
```

Siehe auch

ClrArray, High, StrSort

4.3.7.5 Redim Prozedur

Syntax

```
Redim(var Array; Rank1, Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8,
Rank9, Rank10: Integer)
```

Kategorie

Basisfunktion

Erklärung

Dimensioniert das angegebene Array neu. Dabei kann sowohl die Dimension als auch der Rang (Nummer des höchsten erlaubten Index in einer Dimension) geändert werden. Der Inhalt des Arrays bleibt bestehen. Alle Rang-Parameter sind optional außer der erste.

Beispiel

Das folgende Code-Stück gibt den Wert 20 im Hinweisfenster auf.

```
vardef a: Real[10, 10];
Redim(a, 20, 20, 20);
Trace(Str(High(3, a)))
```

Siehe auch

Variablen, High

4.3.7.6 StrSort Prozedur

Syntax

```
StrSort(List: String[]; Index: Integer): Integer
```

Kategorie

Basisfunktion

Erklärung

Bei der Liste muss es sich um ein eindimensionales String-Array handeln. Dieses wird bis zum durch die Zahl festgelegten Index alphabetisch sortiert. Die möglichen Resultate der Funktion sind:

0	alles in Ordnung
99	Speichermangel

Beispiel

Eine Textdatei wird zeilenweise gelesen, sortiert und dann wieder zurückgeschrieben.

```
vardef Zeilen: String[1000]
vardef i, t, n: Integer;
if t := Reset("EXTERN.TXT")
   i := 0
   while not Eot(t)
      Zeilen[i] := ReadLn(t)
      i := i+1
   end
   Close(t)
   n := i - 1
   StrSort(Zeilen,n)
   t := Rewrite("EXTERN.TXT")
```

```
    nLoop(i, n, WriteLn(t, Zeilen[i]))
    Close(t)
end
```

Siehe auch

InArray

4.3.8 Datum und Uhrzeit

4.3.8.1 Datum und Uhrzeit

Die folgenden Prozeduren zur Behandlung von Datum und Uhrzeit stehen zur Verfügung:

Englischer Name	Deutscher Name	Kurzbeschreibung
DateStr	Datum	Datum eines Zeitstempels als String
DateTimeStr	DateTimeStr	Datum und Uhrzeit eines Zeitstempels als String
DateTimeVal	DateTimeVal	Zeitstempel für einen DatumZeit-String
Day	Tag	Tages-Anteil aus einem Datum
DayOfWeek	WoTag	Nummer des Wochentags
Hour	Stunde	Stunden-Anteil aus einer Uhrzeit
LocalToUtc	LocalToUtc	UTC-Zeit aus einer lokalen Zeit
MakeDate	MachDatum	Zeitstempel für ein Datum ohne Uhrzeit
MakeDateTime	MachZeitstempel	Zeitstempel für ein Datum mit Uhrzeit
MakeTime	MachZeit	Zeitstempel für eine Uhrzeit ohne Datum
Millisecond	Millisekunde	Millisekunden-Anteil aus einer Uhrzeit
Minute	Minute	Minuten-Anteil aus einer Uhrzeit
Month	Monat	Monats-Anteil aus einem Datum
Now	Now	aktuelle Uhrzeit
Second	Sekunde	Sekunden-Anteil aus einer Uhrzeit
TimeStr	Zeit	Zeitstempel als Uhrzeit
Today	Today	das aktuelle Datum
UtcToLocal	UtcInLokal	lokale Zeit aus einer UTC-Zeit
Week	Woche	Kalenderwoche zu einem Zeitstempel
WeekDayNo	WeekDayNo	Nummer des Wochentags
Year	Jahr	Jahres-Anteil aus einem Datum

Falls bei Datumsangaben das Jahrhundert weggelassen ist, gelten diese Regeln:

- Bei Jahreszahleingaben kleiner "30", ergänzt T*urboDB Studio* diese zum Jahre "20xx".
- Eingaben von "30" bis "99" werden automatisch zu "19xx" ergänzt.

4.3.8.2 DateStr Prozedur

Syntax

```
DateStr(Value: DateTime): String
Datum(Wert: DateTime): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt eine Datums-Angabe in eine formatierte Zeichenkette um. Das Format hängt vom letzten Aufruf von *SetNumberFormats* auf und kann von den Rechnereinstellungen abhängig sein. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit" sowie in "Datum und Uhrzeit formatieren".

Beispiele

Am 26.11.1997 liefert der folgende Aufruf den 10.12.1997:

```
DateStr(Today + 14) -> 10.12.1997
DateStr(Geboren)[1, 6] + Str(Year(Today) + 1) -> Geburtstag im nächsten Jahr
als Zeichenkette
```

Siehe auch

TimeStr, DateTimeStr, SetNumberFormats, Datum und Uhrzeit

4.3.8.3 DateTimeStr Prozedur

Syntax

```
DateTimeStr(Value: DateTime; Precision: Integer): String
```

Kategorie

Basisfunktion

Erklärung

Wandelt eine DatumZeit-Angabe in eine formatierte Zeichenkette um. Das Format hängt vom letzten Aufruf von *SetNumberFormats* ab und kann von den Rechnereinstellungen abhängig sein. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit" sowie in "Datum und Uhrzeit formatieren".

Der Parameter *Precision* bestimmt die Genauigkeit der Zeichenkette:

2	Minuten
3	Sekunden
4	Millisekunden

Beispiele

Am 26.11.1997 um 13:25:36 liefert der folgende Aufruf:

```
DateTimeStr(CombineDateTime(Today, Now), 3) -> 26.11.1997 13:25:36
```

Siehe auch

TimeStr, DateTimeVal, SetNumberFormats, Datum und Uhrzeit

4.3.8.4 DateTimeVal Prozedur

Syntax

```
DateTimeVal(Str: String): DateTime
```

Kategorie

Basisfunktion

Beschreibung

Berechnet den Zeitstempel aus einer Zeitangabe im String. Falls der String keine gültige Zeitangabe enthält, wird ein Laufzeitfehler ausgelöst. Bei der Berechnung wird in der folgenden Reihenfolge geprüft:

1. Stimmt der String mit dem aktuellen lokalen Datumsformat überein, wie es auf dem Computer gesetzt ist bzw. mit *SetNumberFormats* definiert
2. Stimmt der String mit einem der Standard-Zeitformate von *TurboDB* überein, dem *TurboDB* -Format DD.MM.YYYY HH:NN:SS,LLLL, dem amerikanischen Format M/D/YY hh:NN:SS.LLLL oder dem deutschen Format DD.MM.YY HH:NN:SS,LLLL.

Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit".

Beispiel

```
DateTimeVal('12:00:00')         -> 12:00:00
DateTimeVal('8.12.2003 20:34') -> 08.12.2003 20:34:00
```

Siehe auch

DateTimeStr, MakeDate, MakeDateTime, Datum und Uhrzeit, Val

4.3.8.5 Day Prozedur

Syntax

```
Day(Value: DateTime): Integer
Tag(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert die Tagesnummer innerhalb des Monats.

Beispiel

Am 26.11.1997 liefert der folgende Aufruf 10.

```
Day(Today + 14) -> 10
```

Siehe auch

DayOfWeek, Month, Year, Datum und Uhrzeit

4.3.8.6 DayOfWeek Prozedur

Syntax

```
DayOfWeek(Value: DateTime): String
WoTag(Wert: DateTime): String
```

Kategorie

Basisfunktion

Erklärung

Zahl als Datum interpretieren und Wochentag ermitteln.

Beispiel

Am 12.Mai 1997 liefert der folgende Aufruf die Zeichenkette "Montag":

```
DayOfWeek(Today)
```

Hier noch ein kleines Programm, das nähere Angaben zu einem Geburtstag ausgibt (Die Anweisungen nach *Message* müssen natürlich in einer Zeile stehen):

```
procedure Geburtstags-Info
    vardef Geburtsdatum: REAL;
    Input("Geben Sie ihr Geburtsdatum ein");
    Geburtsdatum := Val(T-Eingabe);
    Message("Sie sind an einem " + DayOfWeek(Geburtsdatum) + " in der " +
Str(Week(Geburtsdatum)) + ". Kalenderwoche geboren");
endproc
```

Siehe auch

Day, Month, Week, Year, WeekDayNo, Datum und Uhrzeit

4.3.8.7 FormatType Aufzählung

Syntax

```
FormatType = (Local, TurboDB, German, English, International)
```

Kategorie

Basisfunktion

Werte

Bezeichner	Wert	Bedeutung
Local	-1	Aktuelle Windows-Einstellungen verwenden
TurboDB	0	TurboDB, Punkt als Dezimaltrennzeichen, Datum in der Form TT.MM.YYYY, Zeit in der Form HH:MM:SS
German	1	Deutsch, wie TDB nur wird das Jahr zweistellig ausgegeben und ein Komma als Dezimaltrennzeichen verwendet.
English	2	Englisch, der Punkt dient als Dezimalzeichen, Datum in der Form

		MM/DD/YY, die Zeit im 12-Stunden-System mit am und pm: HH:MM am/pm
International	3	International, das Datum als YYYY-MM-DD die Zeit wie bei Deutsch aber mit Komma für die Millisekunden.

Beschreibung

Die Aufzählungswerte legen fest, wie Zahlen, Daten und Uhrzeiten als String formatiert werden können.

Siehe auch

SetNumberFormats Prozedur

4.3.8.8 Hour Prozedur

Syntax

```
Hour(Value: DateTime): Integer
Stunde(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert die Stundennummer innerhalb des Tages.

Beispiel

```
Hour(MakeTime(4, 30, 3, 0)) -> 4
```

Siehe auch

Minute, Second, Millisecond, Datum und Uhrzeit

4.3.8.9 LocalToUtc Prozedur

Syntax

```
LocalToUtc(Local: DateTime): DateTime
LokalInUtc(Lokal: DateTime): DateTime
```

Kategorie

Basisfunktion

Beschreibung

Berechnet die UTC-Zeit aus der Lokalzeit. Der Parameter Local muss mindestens der 1.1.1602 sein, sonst wird ein Laufzeitfehler ausgelöst.

Hintergrund

UTC (Universal Time Coordinates) sind eine Zeitangabe die unabhängig von der Zeitzone und den aktuellen Einstellungen für Sommer/Winterzeit ist. Sie war früher unter dem Namen Greenwich-Zeit bekannt. Zum Speichern von Zeitpunkten in einer Datenbank eignet sich dieses Zeitformat besonders, weil die Datenbank dann auch in Ländern mit anderer Zeitzone noch korrekt ist.

Beispiele

```
TimeStr(LocalToUtc(MakeTime(13, 23, 12, 0))) -> 12:23:12 (in Deutschland zur
Winterzeit)
TimeStr(LocalToUtc(MakeTime(13, 23, 12, 0))) -> 11:23:12 (in Deutschland zur
Sommerzeit)
DateTimeStr(LocalToUtc(MakeDateTime(2003, 12, 8, 0, 35, 10, 350))) ->
7.12.2003 23:35:10.350 (in Deutschland zur Winterzeit)
TimeStr(LocalToUtc(MakeTime(14, 0, 0, 0))) -> 20:0:0 (in Zentralamerika zur
Winterzeit)
```

Siehe auch

UtcToLocal, Datum und Uhrzeit

4.3.8.10 MakeDate Prozedur

Syntax

```
MakeDate(Year, Month, Day: Integer): Date
MachDatum(Jahr, Monat, Tag: Integer): Date
```

Kategorie

Basisfunktion

Beschreibung

Berechnet den Zeitstempel aus den Einzelangaben. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit".

Beispiel

```
MakeDate(2002, 3, 4)           ->  4.3.2002
MakeDate(2003, 3, 34)          ->  Fehler
```

Siehe auch

MakeTime, MakeDateTime, Datum und Uhrzeit

4.3.8.11 MakeDateTime Prozedur

Syntax

```
MakeDateTime(Year, Month, Day, Hour, Minute, Second, Millisecond: Integer):
DateTime
MachZeitstempel(Jahr, Monat, Tag, Stunde, Minute, Sekunde, Millisekunde:
Integer): DateTime
```

Kategorie

Basisfunktion

Beschreibung

Berechnet den Zeitstempel aus den Einzelangaben. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit".

Beispiel

```
MakeDateTime(2002, 3, 4, 12, 0, 0, 0) -> 4.3.2002_00:00:00
```

Siehe auch

MakeDate, MakeTime, Datum und Uhrzeit

4.3.8.12 MakeTime Prozedur

Syntax

```
MakeTime(Hour, Minute, Second, Millisecond: Integer): Time
MachZeit(Stunde, Minute, Sekunde, Millisekunde: Integer): Time
```

Kategorie

Basisfunktion

Beschreibung

Berechnet den Zeitstempel aus den Einzelangaben. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit".

Beispiel

```
MakeTime(12, 0, 0, 0)          ->  12:00:00.000
MakeTime(18, 0, 0, 0)          ->  18:00:00.000
MakeTime(23, 59, 59, 999)->  23:59:59.999
```

Siehe auch

MakeDate, MakeDateTime, Datum und Uhrzeit

4.3.8.13 Millisecond Prozedur

Syntax

```
Millisecond(Value: DateTime): Integer
Millisekunde(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert die Millisekunde innerhalb der Sekunde.

Beispiel

Um 18:25:46.345 liefert

```
Millisecond(Now) -> 345
```

Siehe auch

Hour, Minute, Second, Datum und Uhrzeit

4.3.8.14 Minute Prozedur

Syntax

```
Minute(Value: DateTime): Integer
Minute(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert die Minute innerhalb der Stunde.

Beispiel

Um 18:25:46 liefert

```
Minute(Now) -> 25
```

Siehe auch

Hour, Second, Millisecond, Datum und Uhrzeit

4.3.8.15 Month Prozedur

Syntax

```
Month(Value: DateTime): Integer
Monat(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Interpretiert die Zahl als Datum und liefert die Monatsnummer.

Beispiel

Wenn heute der 1.12.2002 ist dann liefert

```
Month(Today)  ->  12
```

Siehe auch

Day, DayOfWeek, Week, Year, Datum und Uhrzeit

4.3.8.16 Now Prozedur

Syntax

```
Now: Time
```

Kategorie

Basisfunktion

Erklärung

Liefert den aktuellen Zeitpunkt. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit".

Beispiel

```
Message("Die aktuelle Uhrzeit: " + TimeStr(Now))
```

Siehe auch

Hour, Minute, Second, Today, TimeStr, Datum und Uhrzeit

4.3.8.17 Second Prozedur

Syntax

```
Second(Value: DateTime): Integer
Sekunde(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Liefert die Sekunde innerhalb der Minute.

Beispiel

Um 18:25:46 liefert

```
Second(Now) -> 46
```

Siehe auch

Hour, Minute, Millisecond, Datum und Uhrzeit

4.3.8.18 SetNumberFormats Prozedur

Syntax

```
SetNumberFormats(Numerical, Date, Time: FormatType)
```

Kategorie

Basisfunktion

Parameter

Name	Bedeutung
Numerical	Format für Zahlendarstellung
Date	Format für Datumsformatierung
Time	Format für Zeitformatierung

Erklärung

Mit dieser Funktion kann das Ausgabeformat von Fließkommazahlen, Datum und Zeit gesetzt werden, welches unter anderem bei den Funktionen Str, DateStr, TimeStr und DateTimeStr benutzt wird. Die möglichen Werte sind bei der Aufzählung *FormatType* beschrieben. Die Einstellung bleibt für die gesamte Laufzeit des Programms erhalten. Soll in einer Anwendung das Format dauerhaft umgestellt werden, empfiehlt sich der Aufruf von *SetNumberFormats* in *OnOpenProject.*

Beispiel

```
procedure DateAndTimeFormats
  vardef I: Integer;
  for I := 0 to 3
    SetNumberFormats(I, I, I);
    Message(DateStr(8.12.2004) + '    ' + TimeStr(8:12:32, 4));
  next
endproc
```

Diese Prozedur liefert die folgende Ausgabe:

```
08.12.20 08:12:32.000
04
```

```
8.12.04  8:12:32,000
12/8/04  8:12:32.000 am
2004-12- 8:12:32.000
08
```

4.3.8.19 TimeStr Prozedur

Syntax

```
TimeStr(DateTime: DateTime [; Precision: Integer]): String
Zeit(Zeitstempel: DateTime [; Genauigkeit: Integer]): String
```

Kategorie

Basisfunktion

Erklärung

Interpretiert die Zahl als Zeitstempel und wandelt diese in eine Zeichenkette um. Ein etwaiger Datumsanteil wird dabei ignoriert. Der Parameter *Genauigkeit* bestimmt, wie genau die Zeit angezeigt wird:

2 Stunden und Minuten

3 Stunden bis Sekunden

4 Stunden bis Millisekunden

Das Format richtet sich nach dem letzten Aufruf von *SetNumberFormats* und kann von den Rechnereinstellungen abhängen. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit" und in "Datum und Uhrzeit formatieren".

Hinweis

In VDP 3 und früher war das Argument von *TimeStr* kein Zeitstempel sondern eine reine Zeitangabe, also die Anzahl der Minuten seit Mitternacht als Real-Zahl. Wenn man jetzt *TimeStr* mit einer Real-Zahl wie 720,0 aufruft, ist das Ergebnis nicht mehr 12:00 sondern 0:00, weil die Zahl als Datum mit Uhrzeit und nicht als Minuten nach Mitternacht interpretiert wird. Den gewünschten Wert erhält man am besten dadurch, dass man eine Zeit-Variable vom Typ Time verwendet. Allerdings ist auch eine Typ-Konvertierung möglich: *TimeStr(V as Time)*.

Beispiel

```
TimeStr(Now)      ->  "22:57"
TimeStr(Now, 3)   ->  "10:57:34"
TimeStr(Now, 4)   ->  "10:57:34,120"
```

Wenn das Format zuvor anders eingestellt wird:

```
SetNumberFormats(2, 2, 2);
TimeStr(Now)      ->  "10:57 pm"
TimeStr(Now, 3)   ->  "10:57:34 pm"
TimeStr(Now, 4)   ->  "10:57:34.120 pm"
```

Siehe auch

DateStr, DateTimeStr, SetNumberFormats, Datum und Uhrzeit

4.3.8.20 Today Prozedur

Syntax

```
Today: Date
```

Kategorie

Basisfunktion

Erklärung

Liefert eine Zahl für das aktuelle Datum. Diese Zahl kann zum Berechnen von Tagesdifferenzen verwendet oder z.B. mit der Funktion *DateStr* in einen String mit dem lesbaren Datum umgewandelt werden. In der Kombination mit *Now* erhalten Sie einen Wert für den aktuellen Zeitpunkt inklusive Datum und Uhrzeit. Informationen zur Darstellung von Daten und Zeiten finden Sie in "Datum und Uhrzeit".

Beispiel

```
Frist := Today - RechnungsDatum
```

```
DateStr(Today) -> 26.5.2003 (Wert als Beispiel)
DateTimeStr(Today + Now) -> 1.2.2004 09:26 (Wert als Beispiel)
```

Siehe auch

Now, Datum und Uhrzeit

4.3.8.21 UtcToLocal Prozedur

Syntax

```
UtcToLocal(Utc: DateTime): DateTime
UtcInLokal(Lokal: DateTime): DateTime
```

Kategorie

Basisfunktion

Beschreibung

Berechnet die Lokalzeit aus der UTC-Zeit. Der Parameter Local muss mindestens der 1.1.1602 sein, sonst wird ein Laufzeitfehler ausgelöst.

Eine Erläuterung zu UTC finden Sie in "LocalToUtc".

Beispiele

Finden Sie unter "LocalToUtc".

Siehe auch

LocalToUtc, Datum und Uhrzeit

4.3.8.22 Week Prozedur

Syntax

```
Week(Value: DateTime): Integer
Woche(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Interpretiert Zahl als Datum und berechnet daraus die Kalenderwoche.

Achtung

In älteren Versionen verhielt sich Week nicht ganz normgerecht und berechnete in manchen Fälle eine Kalenderwoche 0. Ab TurboDB Studio arbeitet *Week* ISO-konform.

Beispiel

```
Week(17.5.1994) -> 20
```

Siehe auch

Day, DayOfWeek, Month, Year, Datum und Uhrzeit

4.3.8.23 WeekDayNo Prozedur

Syntax

```
WeekDayNo(Value: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Berechnet eine Nummer für den Wochentag des angegebenen Datums. Dabei ist Montag = 1, Dienstag = 2 usw. bis Sonntag = 7.

Beispiel

```
WeekDayNo(28.6.2005) -> 2
```

Siehe auch

DayOfWeek, Datum und Uhrzeit

4.3.8.24 Year Prozedur

Syntax

```
Year(Value: DateTime): Integer
Jahr(Wert: DateTime): Integer
```

Kategorie

Basisfunktion

Erklärung

Interpretiert Zahl als Datum und berechnet daraus die Jahreszahl.

Beispiel

```
Year(1.12.94)  -> 1994
```

Siehe auch

Day, Month, Datum und Uhrzeit

4.4 Datenbank-Befehle

4.4.1 Datenbank-Befehle

Datenbank-Befehle sind interne Funktionen, die auch ohne die Oberfläche von TurboDB Studio auf einer Datenbank ausgeführt werden können. Wenn eine Oberfläche vorhanden ist, ist der Effekt eines Datenbank-Befehls nicht sichtbar bis eine der Funktionen *Refresh* oder *Attach* aufgerufen wird. Mehr Informationen zum Unterschied zwischen Datenbank-Programmierung und Oberflächen-Programmierung finden Sie im Abschnitt "Datenbank-Programmierung und Oberflächenprogrammierung".

Datenbank-Befehle haben praktisch immer einen Tabellenverweis als erstes Argument. Dies kann einfach ein Tabellenname sein, wie in *ReadNext(KUNDEN)* oder aber eine Tabellennummer, die zum Beispiel von *OpenDb* oder *FindTable* zurückgegeben wurde.

Datenbank-Befehle sollten Sie dann einsetzen, wenn Sie umfangreiche Operationen auf den Datenbank-Tabelle ausführen wollen, wie z.B. viele Datensätze neu eintragen oder ändern. Ein anderes Einsatzfeld ist die Vorbereitung von Datenmengen für die Anzeige im Datenfenster. Statt mit Oberflächenfunktionen erreichen Sie mit Datenbank-Befehlen eine schnellere und flimmerfrei Anzeige. Um die Datenmenge tatsächlich im Datenfenster zu sehen, müssen Sie abschließend ein *Attach* im gewünschten Datenfenster aufrufen.

4.4.2 Tabellen

4.4.2.1 Tabellen-Prozeduren

Unter dem Begriff Tabellen-Funktionen werden alle diejenigen Prozeduren und Funktionen zusammengefaßt, die Datentabellen direkt manipulieren. In *TurboDB* Studio ist jedes Datenfenster mit einer oder mehreren Tabellen verknüpft. (Mehrere Tabellen kommen z.B. bei eingebetteten Tabellen ins Spiel.) Im Gegensatz zu den Oberflächen-Funktionen, welche oft auf das Datenfenster wirken und über diesen Umweg unter Umständen auch die Tabelle verändern, arbeiten Tabellen-Funktionen direkt auf den Tabellen. Daraus folgt, dass die Wirkung einer Prozedur wie z.B. *ReadRec* nicht sofort sichtbar wird. Dadurch eignen sich Tabellen-Funktionen zur Durchführung einer komplexen Tabellenoperation aus vielen Einzelbefehlen. Erst wenn die gesamte Operation beendet ist, wird dann mit der Oberflächen-Funktion *Attach* die Übereinstimmung zwischen Datenfenster und Tabelle wiederhergestellt.

Access	Zugriff setzen
AndMarks	Markierungsliste mit aktuellen Markierungen UND-verknüpfen
BlobSize	Ermittelt die Größe eines BLOB
CheckMemos	Überprüfen der Memoeinträge einer Tabelle
ClearDat	Alle Einträge in einer Tabelle löschen
CloseDb	Tabelle schließen
CopyMemo	Kopiert ein Memo in eine Textdatei

DatToDbf	Konvertiert eine TDB-Tabelle in eine dBase-Tabelle
DBDir	Verzeichnis einer Tabellen-Datei
DbfToDat	Konvertiert eine dBase-Tabelle in eine TDB-Tabelle
DBName	Name der Tabellen-Datei
DelMark	Löschen einer internen Markierung
DelMarks	Alle internen Markierungen löschen
DelRec	Aktuellen Datensatz löschen
EditOff	Satzsperre aufheben
EditOn	Satzsperre aktivieren
EmbedBlob	Lädt eine Bild- oder Klangdatei in ein Blobfeld
Exists	Nach verknüpften Datensätzen suchen, die eine Selektion erfüllen.
FileMode	Rechte an einer Tabelle ermitteln
FileNo	Ermittelt die aktuelle Primärdatei
FileSize	Liefert Anzahl der Datensätze einer Tabelle
FindRec	Datensatz über Index suchen
FirstRec	Nummer des ersten Datensatzes
Flush	Tabellen auf Datenträger schreiben
FSum	Summenbildung über Tabellenspalten
GenIndex	Index für eine Tabelle erstellen
GetField	Inhalt eines Datenfeldes
GetLinkedFile	Ermittelt den Pfad zu einem verknüpften BLOB
GetMarks	Markierungen speichern
GetRec	Datensatz in Puffervariable schreiben
GetType	Datentyp eines Feldes
ImportODBC	Import von Datensätzen aus ener ODBC-Datenquelle
IndDef	Indexbeschreibung abfragen
IndName	Name eines Index ermitteln
IndNo	Nummer eines Index ermitteln
IsMark	Prüft, ob aktueller Datensatz im aktiven Datenfenster markiert.
Label	Name einer Tabellenspalte ermitteln
LabelNo	Spaltennummer eines Feldes ermitteln
LastRec	Satznummer des letzten Datensatzes
Link	Komplette Abarbeitung aller verknüpften Datensätze
LinkBlob	Verknüpft eine Bild-Datei mit einem Bild-Feld
LinkCount	Anzahl verknüpfter Datensätze zählen
LinkSum	Summe über verknüpfte Datensätze bilden
Lock	Totalsperre einrichten
MarkRel	Verknüpfte Datensätze markieren
MarkTable	Volltextsuche durchführen
MaxFile	Anzahl der offenen Tabellen
MaxLabel	Anzahl der Spalten in einer Tabelle
MemoLen	Länge eines Memos ermitteln
MemoStr	Die ersten 255 Zeichen eines Memos
Memo2HTML	Memo nach HTML konvertieren
NetUsers	Anzahl der Tabellen-Benutzer
NextRec	Nummer des nächsten Datensatzes
NMarks	Liefert Anzahl der markierten Datensätze
NewTable	Neue Tabelle erzeugen und Struktur übernehmen

NotMarks	Markierungsliste mit aktuellen Markierungen UND NICHT-verknüpfen
OpenDb	Tabelle öffnen
PrevRec	Nummer des vorherigen Datensatzes
PrimFile	Stellt Primärtabelle um
PutMarks	Markierungsliste mit aktuellen Markierungen ODER-verknüpfen
PutRec	Datensatz in Tabelle schreiben
ReadMemo	Memo lesen
ReadRec	Datensatz lesen
RecNo	Aktuelle physikalische Satznummer des aktiven Datenfensters ermitteln
RegenAll	Alle Indexe wiederherstellen
RegenInd	Index wiederherstellen
RelIndex	Beschleunigt Zugriff via Relationsfeld
RollBack	Ursprünglichen Tabellenzustand wiederherstellen
ScanRec	Aktualisieren eines bestehenden Volltextindex
ScanRecs	Anlegen eines Volltextindex
SetAuto	Autonummer setzen
SetField	Datenfeld verändern
SetMark	Setzen einer internen Markierung
SetRecord	Datensatz von einer Tabelle in eine andere importieren
SortMark	(interne) Markierungen sortieren
TransOff	Transaktionsmodus beenden
TransOn	Transaktionsmodus beginnen
Unlock	Sperre aufheben
WriteRec	Datensatz schreiben

4.4.2.2 CheckMemos Prozedur

Syntax

```
CheckMemos(Tabelle: Integer): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

CheckMemos prüft die Memo-Datei der Tabelle auf evtl. Fehler.

Der Rückgabewert ist:

1	Alles ok, aber manche Blöcke sind verwaist. Restrukturieren der Tabelle würde die Memo-Datei verkleinern
0	Alles ok
-1	Die Memodatei ist abgeschnitten, wahrscheinlich fehlen Daten. Restrukturieren der Tabelle stellt zumindest wieder einen fehlerfreien Zustand her.
-2	Die Liste der freien Blöcke überschneidet sich. Tabelle restrukturieren, kein Datenverlust.
-3	Mindestens ein Datensatz enthält einen ungültigen Memo-Verweis. Tabelle restrukturieren. Wahrscheinlich wurden schon Daten verloren.
-4	Mindestens ein Memo überschneidt sich mit der Freiliste. Tabelle dringend restrukturieren, um Datenverlust zu verhindern.
-5	Die Memos überschneiden sich. Sie haben doppelte Memos in der Tabelle.Restrukturieren Sie die Tabelle, dann können Sie die defekten Memos wieder korrigieren.

Falls die angegebene Tabelle über kein Memo verfügt, wird 0 zurückgegeben.

Achtung

Falls Sie einen negativen Rückgabewert erhalten, müssen Sie auf jeden Fall die Tabelle restrukturieren, um wieder einen fehlerfreien Zustand herzustellen. Sichern Sie aber in jedem Fall zuvor alle Dateien der Tabelle, damit ggf. verlorene Daten wiederhergestellt werden können und prüfen Sie nach dem Restrukturieren, ob Ihre Daten in Ordnung sind.

Beispiel

Dieses Routine prüft alle Memos eines Projektes mit 11 Tabellen:

```
const TableNum = 11;

procedure CheckAllMemos;
    vardef TableNo, Result: Integer;
    for TableNo := 1 to TableNum
       Result := CheckMemos(TableNo);
       if Result = 1
          Message("Die Tabelle " + DbName(TableNo) + " sollte mal restrukturiert
werden");
       elsif Result < 0
          Message("Die Tabelle " + DbName(TableNo) + " muss restrukturiert
werden. Datenverlust möglich.");
       end;
    next;
endproc
```

4.4.2.3 ClearDat Prozedur

Syntax

```
ClearDat(Tabelle: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Löscht ohne weitere Rückfrage sämtliche Datensätze aus der Tabelle.

Beispiel

```
if Message("Sollen wirklich alle Datensätze der Tabelle 'KFZ' gelöscht
werden?", "Bestätigen", 3) = 6
    ClearDat(KFZ)
end
```

Siehe auch

SetAuto

4.4.2.4 CloseDb Prozedur

Syntax

```
CloseDb(Tabellennummer: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Entfernt die Tabelle aus der Datenbank und schließt sie dabei physikalisch. Tabellennummer ist zum Beispiel die von *OpenDb* oder *FindTable* zurückgegebene. Falls Tabellennummer negativ ist, entfernt *CloseDb* alle unbenutzten Tabellen aus der Datenbank. Dies schließt mit *OpenDb* geöffnete genauso ein, wie temporäre Tabellen, die im Rahmen eines SQL-Befehls angelegt wurden.

Die Funktion wird selten benötigt, weil die mit *OpenDb* geöffnete Tabelle automatisch geschlossen wird. Sie kann aber dann sinnvoll sein, wenn man auf die Datei einer Tabelle mit normalen Dateifunktionen zugreifen möchte.

Die Tabelle kann nur dann entfernt werden, wenn sie nicht gerade in Benutzung ist, d.h. kein Cursor dafür existiert.

Das Ergebnis ist immer 0. Die Systemvariable Fehler gibt über etwaiges Fehlschlagen Auskunft.

Siehe auch

OpenDb, FindTable

4.4.2.5 CountRecs Prozedur

Syntax

```
CountRecs(Suchbedingung): Integer
```

oder

```
LinkCount(Suchbedingung): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Zählt die Anzahl der Datensätze, die der Suchbedingung entsprechen. Wenn man *CountRecs* auf die Primärtabelle anwendet, wird diese vollständig durchsucht (neu seit *TurboDB Studio*). Angewendet auf eine andere Tabelle, werden die mit dem aktuellen Datensatz der Primärtabelle verknüpften Datensätze gezählt. Diese Verknüpfung kann wie in *TurboDB* üblich mittels Koppelfeld, Relationsfeld oder statischem Link definiert sein.

Beispiel

Die folgende Prozedur wird aus einem Datenfenster der KFZ-Tabelle aus aufgerufen und ermittelt wieviele Kunden mit dem aktuellen Datensatz der KFZ-Tabelle verknüpft sind. (Die KUNDEN-Tabelle hat ein Koppelfeld auf KFZ.) Das Kommando primtableis ist hier eigentlich überflüssig, weil beim einem Aufruf aus einem Formular, die Primärtabelle automatisch auf die Tabelle des Formulars gesetzt wird.

```
procedure Zeige_AnzahlKunden_zu_Fahzeug;
    primtableis KFZ
    Message(Str(CountRecs(KUNDEN)))
endproc
```

Siehe auch

Link, LinkSum, Statistik-Funktionen

4.4.2.6 DatToDbf Prozedur

Syntax

```
DatToDbf(Tabelle: Integer; DbfDateiName: String): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

DatToDbf exportiert die Datensätze aus der Tabelle in eine neue Dbf-Tabelle. Die *TurboDB* Tabelle muss im aktuellen Projekt enthalten sein. Der Rückgabewert ist 0 falls alles gutgegangen ist, sonst < 0.

Beispiel

```
if DatToDbf(KUNDEN, "C:\dBase\Kunden.dbf") < 0
    Message("Fehler bei Konvertierung")
end
```

Siehe auch

DbfToDat

4.4.2.7 DBDir Prozedur

Syntax

```
DBDir(Tabelle: Integer): String;
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert das Verzeichnis einer Tabellen-Datei.

Beispiel

```
Message(DBDir(KFZ)); -> z.B: C:\Daten
```

Siehe auch

DBName, BaseDir, PrivDir

4.4.2.8 DbfToDat Prozedur

Syntax

```
DbfToDat(DbfDateiName, DatDateiName: String): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

Konvertiert die dBase-Tabelle in eine neue TDB-Tabelle. Der Rückgabewert ist 0 bei Erfolg, sonst ein negativer Fehlercode.

Beispiel

```
if DbfToDat("q:\vdpdev.250\testsuit\import\dBase\excel.dbf", "s:\test.dat") <
0
    Message("Fehler bei der Konvertierung")
end
```

Siehe auch

DatToDbf

4.4.2.9 DBName

Syntax

```
DBName(Tabelle: Integer): String;
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Name einer Tabellen-Datei. Zusammen mit *DBDir* ergibt sich der vollständige Pfadnamen der Tabellen-Datei.

Beispiel

```
Message(DBName(KUNDEN)); -> KUNDEN.DAT
```

Siehe auch

DBDir, IndName

4.4.2.10 DelTable Prozedur

Syntax:

```
DelTable(TableName: String; Password: String; Code: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Löscht eine Datenbank-Tabelle.

Achtung:

Damit sind alle Daten der Tabelle unwiederruflich vernichtet. Dieser Befehl kann nicht rückgängig gemacht werden.

Siehe auch:

NewTable, OpenDb, CloseDb

4.4.2.11 EnumStr Prozedur

Syntax

```
EnumStr(Table, FieldNo, EnumValue: Integer): String
EnumStr(Tabelle, FeldNr, EnumWert: Integer): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Text, der zum angegebenen Wert des Aufzählungstyps gehört.Falls die Tabelle oder *FeldNr* ungültig sind, oder wenn *FeldNr* kein Aufzählungsfeld bezeichnet, wird ein Fehler ausgelöst. Wenn *EnumWert* kein gültiger Wert für das Feld ist (entweder zu groß oder zu klein), dann liefert die Funktion einen Leerstring.

Beispiel

Nehmen wir an dass die Tabelle KUNDEN im achten Feld die Aufzählungswerte *männlich* und *weiblich* zulässt.

```
EnumStr(KUNDEN, 8, 1)    ->     'männlich'
EnumStr(KUNDEN, 8, 2)           ->     'weiblich'
EnumStr(KUNDEN, 8, 3)           ->     ''
EnumStr(KUNDEN, 9, 2)           ->     Fehler
```

Siehe auch

EnumVal, GetType

4.4.2.12 EnumVal Prozedur

Syntax

```
EnumVal(Table, FieldNo: Integer; EnumName: String): Integer
EnumVal(Tabelle, FeldNr: Integer; EnumName: String): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den nummerischen Wert, der zum Text eine Aufzählungstyps gehört. Falls die Tabelle oder *FeldNr* ungültig sind, oder wenn *FeldNr* kein Aufzählungsfeld bezeichnet, wird ein Fehler ausgelöst. Wenn *EnumName* kein gültiger Wert für das Feld ist (entweder zu groß oder zu klein), dann liefert die Funktion den Wert 0.

Beispiele

Nehmen wir an, dass die Tabelle KUNDEN im achten Feld die Aufzählungswerte männlich und weiblich zulässt. Dann setzt der folgende Code für a = 8 b auf 1 und zeigt für andere Werte von a eine Meldung an.

```
vardef b: Integer
.EC 1
b := EnumVal(KUNDEN, a, 'männlich');
if Error.Number > 0
   Message(Error.Description);
end
```

Anmerkung

Den aktuellen nummerischen Wert eines Aufzählungsfeldes erhält man auch durch einfache Angabe des Feldnamens. Auch mit Val kann man den Text in einen nummerischen Wert konvertieren, allerdings kann es hier zu Konflikten bei gleichnamen Aufzählungswerten kommen.

Siehe auch

EnumStr, GetType

4.4.2.13 Exists Prozedur

Syntax

```
Exists(Selektion)
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert 1, wenn die Selektion erfüllbar ist, andernfalls 0. Um die Erfüllbarkeit zu prüfen, werden die mit dem aktuellen Satz der Primärtabelle verknüpften Sätze der in der Selektion angesprochen Tabellen gelesen, bis entweder die Selektion zutrifft oder keine weiteren Verknüpfungen mehr gebildet werden können.

Beispiel

(Primärtabelle ist KFZ):

```
Exists(KUNDEN.Name = "Müller")
```

liefert 1, wenn es wenigstens einen Kunden names "Müller" für das aktuelle Auto gibt.

```
NOT Exists(KUNDEN.Name<>"Müller")
```

liefert 1, wenn alle Kunden des aktuellen Autos den Namen "Müller" haben.

Siehe auch

Link, Sub

4.4.2.14 FileMode Prozedur

Syntax

```
FileMode(Tabelle: Integer): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Zugriffsmodus einer Tabelle in Form einer Zeichenkette als Kombination aus den Buchstaben "N", "E", "L" und "I". Dabei gilt:

"N"	Neueingabe, Import und Kopieren von Datensätzen verboten
"E"	Editieren verboten
"L"	Löschen von Datensätzen verboten
"I"	Indizierung verboten

Beispiel

```
if Scan("N", FileMode(KUNDEN))
    Message("Es dürfen keine Daten gelöscht werden!", "Hinweis")
end
```

4.4.2.15 FileNo Prozedur

Syntax

```
FileNo: Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion *FileNo* ermittelt den Handle der aktuellen Primärdatei.

Beispiel

Ausgabe der Größe der aktuellen Primärtabelle.

```
Message('Tabellengröße: ' + Str(FileSize(FileNo)))
```

Siehe auch

FileSize, IndNo

4.4.2.16 FileSize Prozedur

Syntax

```
FileSize(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Anzahl der Datensätze der angegebenen Tabelle.

Beispiel

```
Message("Die Tabelle KUNDEN beinhaltet " + Str(FileSize(KUNDEN)) + "
Einträge", "Meldung");
```

Siehe auch

FileNo

4.4.2.17 Flush Prozedur

Syntax

```
Flush
```

Kategorie

Datenbank-Befehl

Erklärung

Alle offenen Tabellen werden physikalisch geschlossen. Die Verzeichniseinträge werden aktualisiert. Ebenso werden alle gepufferten Indexinformationen zurückgeschrieben und der von den einzelnen Indexseiten belegte Speicherplatz freigegeben. Nach dem Aufruf von *Flush* befindet sich das Projekt damit in einem gesicherten Zustand.

4.4.2.18 FindTable Prozedur

Syntax

```
FindTable(TableName: String): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Tabelle-Nummer für die angegebene Tabelle. Falls die Tabelle nicht offen ist, wird 0 zurückgegeben.

Beispiel

Diese Prozedur prüft, ob eine Tabelle geöffnet ist und öffnet sie, falls das nicht der Fall ist.

```
procedure GetTable(TableName: String): Integer;
  vardef Result: Integer;
  Result := FindTable(TableName);
  if Result = 0
    Result := OpenDB(TableName, '', 0, 15);
  end;
  return Result;
endproc;
```

4.4.2.19 FSum Prozedur

Syntax

```
FSum(Column, Count: Integer): Real;
```

Kategorie

Datenbank-Befehl

Erklärung

Stehen mehrere Spalten innerhalb der Tabellendefinition direkt nebeneinander, liefert *FSum* die Summe dieser Spalten für den aktuellen Datensatz. Alphanummerische Spalten werden einfach verkettet. Die Summenbildung beginnt bei Spalte *Column* und geht über *Count* Spalten.

Beispiel

Bezeichnung	Wert1	Wert2	Wert3

"Silber"	12	13	11
"Gold"	23	21	13

```
ReadRec(Tabelle, 1)
FSum(Wert1, 3)                  ->          36
ReadRec(Tabelle, 2)
FSum(Wert1, 3)                  ->          57
```

Siehe auch

Sum

4.4.2.20 GetLinkedFile Prozedur

Syntax

```
GetLinkedFile(Blob: Feld): String
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion liefert den Pfad zum übergebenen Blob-Feld. Voraussetzung dafür ist natürlich, dass es sich um eine verknüpfte Bild- bzw. Klangdatei handelt. Für eingebettete Blobs wird eine leere Zeichenkette zurückgegeben. In diesem Fall verwenden Sie die Funktion *PlaySound* zum Abspielen des Klanges.

Beispiel

Die Prozedur spielt einen vorhandenen Sound. Ist das Blob leer und der Editier- bzw. Neueingabemodus aktiviert, wird ein Dateidialog zum Verknüpfen einer WAVE-Datei angezeigt.

```
PROCEDURE Hörbeispiel;
      IF Sample
           VarDef BlobPfad: String;
           BlobPfad := GetLinkedFile(Sample);
           IF Length(BlobPfad) > 0
                PlayMedia(GetLinkedFile(Sample));
           ELSE
                PlaySound(Sample);
           END;
      ELSE
           IF GetMode <> 0
                T-Eingabe := '*.WAV';
                IF ChooseFile('Sound laden')
                        IF Upper(RightStr(T-Eingabe, 4)) = '.WAV'
                                    LinkBlob(Sample, T-Eingabe, 3);
                        ELSE
                                    Message( 'Ungültges Dateiformat',
'Fehlerhafte Eingabe');
                        END;
                END;
           END;
      END;
ENDPROC;
```

Siehe auch

BlobSize, MediumPause, MediumSpielen, MediumStop, PlaySound

4.4.2.21 GetType Prozedur

Syntax

```
GetType(Table, FieldNo [, Format]: Integer): String
GetType(Tabelle, Feldnummer [, Format]: Integer): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Typ des Feldes mit der angegebenen Feldnummer. Mit *Format* (optional) kann die maximale Länge des Rückgabestrings bestimmt werden:

1 Nur den reinen Typ
2-4 Längen- bzw. Genauigkeit mit einschließen
5 Optionen mit einschließen

Der Vorgabewert für Format ist 1.

Mögliche Rückgabewerte sind:

A Auswahlfeld
B Byte
C Einzelzeichen
D Datumsfeld
Fnnn Kommazahl mit nnn Nachkommastellen
I Kurzer Integer (16 Bit)
J JaNein
L Koppelfeld
M Memo
N Autonummer
P Bild/Klang
R Relationsfeld
Snnn Zeichenkette mit der maximalen Länge nnn
Znnn Zeitfeld mit Genauigkeit nnn (1 = Stunden, 2 = Minuten, 3 = Sekunden, 4 = Millisekunden, 5 = Mikrosekunden)
G Integer (32 Bit)
T DatumZeit
Wnnn Unicode String mit der maximalen Länge in nnn
Y Unicode Memo
H Langer Integer (64 Bit)
U Guid

Falls die Optionen mit angegeben werden, kommt anschließend ein Komma, gefolgt von der Liste der gesetzten Optionen für das Feld:

N Feld erlaubt null-Werte.

Versionen

Die Angabe der Genauigkeit bei Kommazahlen und Zeitfeldern sowie die Möglichkeit, Optionen mit anzugeben sind neu in *TurboDB Studio*.

Beispiel

```
vardef TypeDef: String;
TypeDef := GetType(IRGENDEINETABELLE, IrgendeinFeld, 5);
if TypeDef[1] = 'I'
   Message('Das Feld ist ein 16-Bit Zahlfeld.');
end;
vardef Options: String;
Options := TypeDef[Scan(',', TypeDef) + 1, 255];
if Scan('0', Options)
   Message('Das Feld erlaubt NULL-Werte.');
end;
```

Siehe auch

Label, EnumStr, EnumVal

4.4.2.22 ImportODBC Prozedur

Syntax

```
ImportODBC(Tabelle: Real; Datenquelle, Benutzername, Passwort, SqlAbfrage:
String): Real;
```

Kategorie

Datenbank-Befehl

Erklärung

Mit dieser Funktion werden Datensätze aus einer ODBC-Datenquelle in eine existierende Tabelle im TurboDB-Format importiert. Die Zuordnung der zu importierenden Felder zu den Feldern der TurboDB-Tabelle erfolgt der Reihe nach. Wenn die Datentypen der Felder der ODBC-Datenquelle und der TurboDB Tabelle nicht kompatibel sind, so wird der Import abgebrochen.

Die Parameter haben die folgende Bedeutung:

Tabelle	Ziel-Tabelle
Datenquelle	ODBC-Datenquelle wie in Systemsteuerung eingetragen
Benutzername	Benutzername für ODBC-Datenquelle
Passwort	Passwort für ODBC-Datenquelle
SqlAbfrage	SQL-Statement für die zu importierenden Daten.

Die Bedeutung der Rückgabewerte:

0	Der Datenimport war erfolgreich.
-1	Die Zieltabelle ist nicht geöffnet.
-2	ODBC ist auf dem Computer nicht installiert.
-3	Die ODBC-Datenquelle existiert nicht.
-4	Das SQL Statement ist fehlerhaft.
-5	Die Datentypen der Felder der Quell- und der Ziel-Tabelle passen nicht.

Beispiel

```
IF ImportODBC(KUNDEN, "PARADOX_KUNDEN", "", "", "SELECT * FROM CLIENTS") < 0
    Message("Fehler in ODBC-Import");
END;
```

Siehe auch

DatensätzeImportieren

4.4.2.23 Label Prozedur

Syntax

```
Label(Tabelle, Feldnummer: Integer, [Komplett: Integer]): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Namen (Feldbezeichner) des Feldes mit der angegebenen *Feldnummer*. Über den Parameter *Komplett* kann man entscheiden, ob bei Koppel- oder Relationsfeldern die angekoppelte Tabelle mit ausgegeben wird.

Beispiel

In der KUNDEN-Tabelle des KFZ-Beispiels liefert Label die folgenden Werte. Das elfte Feld ist das Koppelfeld auf KFZ:

```
Label(KUNDEN, 1)  ->     Name
Label(KUNDEN, 2)  ->     Vorname
Label(KUNDEN, 11) ->     Fahrzeug
Label(KUNDEN, 11, 1) -> Fahrzeug: (KFZ)
```

Siehe auch

LabelNo, GetType

4.4.2.24 LabelNo Prozedur

Syntax

```
LabelNo(Table: Integer; ColumnName: String): Integer
LabelNr(Tabelle: Integer; Feldbezeichner: String): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Feldnummer des Feldes mit der angegebenen Feldbezeichnung. In der DOS-Version hieß diese Funktion *LabelNr*.

Beispiel

Für die Tabelle KUNDEN im KFZ-Beispiel liefert die Funktion *LabelNo* die folgenden Werte:

```
LabelNo(KUNDEN, "Name")              1
LabelNo(KUNDEN, "Vorname")           2
```

Siehe auch

Label

4.4.2.25 LoopRecs Prozedur

Syntax

```
LoopRecs(Suchbedingung)
```

oder

```
Link(Suchbedingung)
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert 1, wenn die Suchbedingung erfüllbar ist, andernfalls 0. Um die Erfüllbarkeit zu prüfen, werden alle mit dem aktuellen Satz der Primärtabelle verknüpften Sätze der in der Selektion angesprochen Tabellen gelesen. Zusätzlich werden die statistischen Funktionen aktualisiert. Falls die Selektion die Primärtabelle betrifft, wird diese komplett gelesen.

LoopRecs ist eine Alternative zu *Sub*/*EndSub* oder einer Schleife mit *FirstRec*/*NextRec*. Im Vergleich zu diesen ist *LoopRecs* erheblich kompakter, weil man die ganze Schleife in einer einzigen Zeile schreiben kann und wird auch deutlich schneller ausgeführt. Dafür bietet *LoopRecs* nicht so viele Möglichkeiten wie die anderen genannten Alternativen.

Anmerkung

Da in Suchbedingungen der Operator and auch durch ein Komma ausgedrückt werden kann, werden mehrstufige Suchbedingungen häufig so geschrieben:

```
LoopRecs(KUNDEN, Name = 'Müller', SetMark(KUNDEN, RecNo(KUNDEN)));
```

um beispielsweise alle verknüpften Datensätze in der KUNDEN-Tabelle zur markieren.

Beispiel

(Primärtabelle ist KFZ):

```
LoopRecs(KUNDEN.Name = "Müller") * COUNT(KUNDEN)
```

liefert die Anzahl der Kunden namens "Müller" des aktuellen Autos.

```
LoopRecs(KFZ) * MEAN(KFZ.Katalysator) * 100
```

liefert den prozentualen Anteil der KFZ mit Katalysator.

Die Funktion kann auch dazu verwendet werden, um nur eine Untermenge der Datensätze der aktuellen Primärdatei anzuzeigen:

```
OpenForm("KUNDEN.Kunden_FORMULAR");
LoopRecs(Name wie "A*" and SetMark(KUNDEN, RecNo(KUNDEN)));
Access(KUNDEN, "Markierung");
```

```
Attach;
ShowRec(-1);
ViewRecs("Alle Kunden deren Namen mit 'A' beginnen");
CloseWnd;
```

Ein weiterer wichtiger Anwendungsfall ist das Markieren angekoppelter Datenssätze. Primärtabelle ist wiederum KFZ, angezeigt werden alle Kunden, die mit dem ausgewählten Auto verknüpft sind:

```
vardef Rec: Integer;
DatensätzeBearbeiten("KFZ.Formular_KFZ")
Rec := DatensatzAuswählen("Selektieren Sie ein Fahrzeug")
if Rec > 0
   DatensätzeBearbeiten("KUNDEN.Formular_KUNDEN")
   PrimFile(KFZ)
   ReadRec(KFZ, Rec)
   LoopRecs(KUNDEN and SetMark(KUNDEN, RecNo(KUNDEN)))
   Access(KUNDEN, "Markierung")
   Attach
end
```

Siehe auch

Access, Attach, Exists, CountRecs, SumRecs, ShowRec, Statistik-Funktionen, ViewRecs

4.4.2.26 MaxFile Prozedur

Syntax

```
MaxFile: Integer
```

Erklärung

Die Anzahl der offenen Tabellen wird zurückgegeben.

Kategorie

Datenbank-Befehl

Beispiel

```
Message("Es sind " + Str(MaxFile) + " Tabellen geöffnet")
```

Siehe auch

MaxLabel

4.4.2.27 MaxLabel Prozedur

Syntax

```
MaxLabel(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion *MaxLabel* ermittelt zur Laufzeit des Programms die Anzahl der Spalten einer Datenbanktabelle. Besonders geeignet ist die Funktion in Abbruchbedingungen von Schleifenkonstruktionen.

Beispiel

Das folgende Makro scheibt alle Feldbezeichner der aktuellen Primärtabelle in eine externe Textdatei.

```
PROCEDURE WriteLabels
      VARDEF t, i : REAL
      t := Reset("EXTERN.TXT")
      WHILE i := i+1 <= MaxLabel(FileNr)
           WriteLn(t,Label(FileNr,i))
      END
      Close(t)
ENDPROC
```

Siehe auch

FileSize, Label, LabelNo

4.4.2.28 NewTable Prozedur

Syntax

```
NewTable(Table: Integer; NewTableName: String[; FirstRecordId, TableLevel:
Integer]): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

NewTable erzeugt eine neue Tabelle mit dem Namen Dateiname, welche die gleiche Struktur hat, wie die angegebene schon geöffnete Tabelle. *NewTableName* ist der vollständige Pfad zur neuen Tabelle inklusive Extension (normalerweise dat). Optional kann bei Tabellen mit Auto-Nummern-Feld der neue Startwert der Nummerierung angegeben werden. Der Vorgabewert ist hier 1. Ebenfalls optional ist die Angabe des Dateiformates. Die möglichen Werte sind:

0	Dateiversion von Tabelle übernehmen (Vorgabe)
1	Format TDB 1.0 (kompatibles Format für TDB DOS und Windows sowie VDP bis Version 2.0
2	Format TDB 4.0 (Standardformat für alle neuen Tabellen ab VDP 3.0)
3	Format TDB 5.0 (Standard für TurboDB und TurboDB Studio)

Das Ergebnis ist Null, wenn die Tabelle erfolgreich erzeugt werden konnte, andernfalls wird ein Fehlercode geliefert.

Neu ab TurboDB Studio 4

Wenn schon eine Datei mit dem angegebenen Namen existiert, wird sie nicht überschrieben, sondern ein Fehler zurückgeliefert. Löschen Sie die Tabelle in diesem Fall zuvor mit *DelTable*.

Beispiel

```
NewTable(KUNDEN, "X:\BACKUP\12121997.DAT");
```

Siehe auch

DelTable

4.4.2.29 OpenDb Prozedur

Syntax

```
OpenDb(Dateiname, Passwort: String; Code, Zugriffsrechte: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Fügt die Tabelle vorübergehend zur Datenbank hinzu. Dabei wird die angegebene Tabelle physikalisch geöffnet.

Dateiname ist die komplette Pfadangabe der Tabelle inklusive Extension, bei Datenbank-Dateien einfach nur der Tabellenname. *Passwort* und *Code* müssen übergeben werden, wenn die Tabelle geschützt ist, ansonsten tragen Sie hier "" und 0 ein. Beim Passwort sind Großbuchstaben anzugeben. Verwenden Sie beispielsweise die Funktion *Upper* zur Umwandlung einer Benutzereingabe in Großbuchstaben. Bei den *Zugriffsrechten* (NELI-Rechten) können Sie in Form von gesetzten Bits bestimmen, welche Aktionen auf der Tabelle erlaubt sein sollen:

Bit-Nr	Bit-Wert	Bedeutung
0	1	**N**eueingabe
1	2	**E**ditieren
2	4	**L**öschen
3	8	**I**ndizieren

Beispiele für den Wert von Zugriffsrechte:

0	Nur lesen
6	Editieren und Löschen erlaubt
7	Neueingabe, Editieren und Löschen erlaubt

8 Indizieren erlaubt

15 Alles erlaubt

Das Ergebnis der Funktion ist die Tabellennummer, wenn die Funktion erfolgreich geöffnet werden konnte. Wenn die Tabelle nicht geöffnet werden kann, wird der Rückgabewert auf 0 gesetzt und ein Laufzeitfehler ausgelöst. Mögliche Fehler sind:

Fehlernummer	Mögliche Ursache
1	Tabellenname falsch angegeben
35	Tabelle ist schon im Projekt eingebunden oder wurde schon mit OpenDb geöffnet. Es kann auch sein, dass eine andere Tabelle mit dem selben Namen schon geöffnet wurde. Denken Sie hier auch an automatisch geöffnete Relationstabellen.

Eine *OpenDb* geöffnete Tabelle wird vom Laufzeitsystem wieder geschlossen, wenn sie nicht mehr benötigt wird. Dies kann schon am Ende der aufrufenden Prozedur sein, oder auch später. Die Anwendung muss sich darum nicht kümmern. Mit *CloseDb* kann man die Tabelle wieder aus der Datenbank entfernen und dabei physikalisch schließen.

Beispiel

Öffnen der unverschlüsselten und nicht passwortgeschützten Tabelle KUNDEN mit allen Rechten.

```
procedure Kundentabelle_öffnen
    vardef TableHdl: Integer;
    ..Fehlerbehandlung selbst übernehmen
    .EC 1
    TableHdl := OpenDb("KUNDEN.DAT", "", 0, 15)
    .EC 0
    IF TableHdl > 0
       ..
       ..hier kann mit der Tabelle gearbeitet werden
       ..
       ..Das Schließen der Tabelle muss nicht unbedingt hier erfolgen,
       ..sollte aber auf keinen Fall vergessen werden
       CloseDb(TableHdl)
    else
       Message("Beim Öffnen der Tabelle KUNDEN ist ein Fehler aufgetreten.
Fehlernummer: " + Str(Fehler))
    end
endproc
```

Öffnen der unverschlüsselten und nicht passwortgeschützten Tabelle KUNDEN, wobei nur Editieren und .Indizieren erlaubt ist

```
TableHdl := OpenDb("KUNDEN.DAT", "", 0, 10)
```

Jetzt ist die Tabelle mit dem Passwort Hugo und dem Schlüssel 1234 gesichert:

```
TableHdl := OpenDb("KUNDEN.DAT", "Hugo", 1234, 10)
```

Siehe auch

FindTable, CloseDb

4.4.2.30 PrimFile Prozedur

Syntax

```
PrimFile(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Setzt die Primärtabelle in einer Prozedur. In Datenbankjobs soll das Kommando *primtableis* benutzt werden. Der Unterschied zwischen diesen beiden Arten besteht darin, dass bei *PrimFile* die neue Primärtabelle zur Laufzeit ausgerechnet und gesetzt wird. Bei *primtableis* liegt die neue Primärtabelle schon beim Übersetzen fest. Datenbankjobs benötigen diese feste Angabe der Primärtabelle.

Beispiel

```
PrimFile(KUNDEN)
```

Siehe auch

FileNo, PrimTableIs

4.4.2.31 RecNo Prozedur

Syntax

```
RecNo(Table: Integer): Integer
```

oder

```
RecNr(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die aktuelle physikalische Satznummer der angegebenen Tabelle. Wenn die Tabelle keinen Datensatz enthält oder gerade eine neuer Datensatz eingegeben wird, dann liefert die Funktion 0.

Beispiel

Alle Kunden deren Name mit "B" beginnt, werden markiert.

```
DelMarks;
LoopRecs(KUNDEN.Name wie "B*", SetMark(KUNDEN, RecNo(KUNDEN)))
```

Siehe auch

FileNo

4.4.2.32 SetAuto Prozedur

Syntax

```
SetAuto(Tabelle, Startwert: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion setzt die nächste zu vergebende Laufende Nummer der *Tabelle* auf den im Parameter *Startwert* übergebenen Wert. Wird 0 als Startwert übergeben, kann die nächste Laufende Nummer abgefragt werden, ohne einen neuen Wert zu setzen.

Die Funktion ist mit Sorgfalt zu verwenden, da durch unsachgemäßen Einsatz die Vergabe doppelter "laufender" Nummer möglich ist.

Beispiel

Das folgende Makro setzt die Autonummern auf die nächste Hunderter-Grenze.

```
procedure NeueNummern
    SetAuto(FileNr, 100*(1+(SetAuto(FileNr, 0)-1) DIV 100))
endproc
```

Siehe auch

ClearDat

4.4.2.33 SetFilter Prozedur

Syntax

```
SetFilter(Tabelle: Integer; UntererWert: String [; ObererWert: String])
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion setzt einen Bereich für die Tabelle ähnlich wie das Filter-Kommando. Im Unterschied zu diesem ist *SetFilter* aber dynamisch und kann die Werte zur Laufzeit festlegen. Mit *SetFilter(<Tabelle>, '')* wird der Filter gelöscht.

Siehe auch

Filter, Access

4.4.2.34 SetMark Prozedur

Syntax

```
SetMark(Tabelle, Satznummer: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Markiert den Satz mit der angegebenen physikalischen Satznummer der Tabelle. Falls der Satz bereits markiert ist, ist die Funktion ohne Wirkung. Das Ergebnis der Funktion ist 1, wenn der Satz markiert werden konnte, andernfalls 0.

Beispiel

Markieren des zweiten Datensatzes der KUNDEN-Tabelle:

```
SetMark(KUNDEN, 2)
```

Siehe auch

AndMarks, DelMark, DelMarks, GetMarks, IsMark, NMarks, NotMarks, PutMarks, SortMark

4.4.2.35 SumRecs Prozedur

Syntax

```
SumRecs(Suchbedingung): Real
```

oder

```
LinkSum(Selektion): Real
```

Kategorie

Datenbank-Befehl

Erklärung

Bildet die Summe über ein Feld aus einer angekoppelten Datei. Dabei werden alle an den aktuellen Datensatz der Primärdatei angekoppelten Datensätze berücksichtigt.

Beispiel

Um in einem Formular die Gesamtsumme über eine Tabellenspalte (hier Anzahl) anzuzeigen, wird ein Formelfeld (Anzeigeelement) mit folgender Formel eingefügt:

```
Str(SumRecs(Anzahl))
```

Siehe auch

LoopRecs, CountRecs, Statistik-Funktionen

4.4.3 Statistik-Funktionen

4.4.3.1 Statistik-Funktionen

Statistik-Funktionen werden hauptsächlich in SQL-Abfragen und in Datenbank-Berichten eingesetzt. Sie werden oft auch Aggregations-Funktionen genannt.

4.4.3.2 aggregates Kommando

Syntax

```
aggregates <Aggregations-Funktion1>, <Aggregations-Funktion2>
```

Kategorie

Datenbank-Befehl

Erklärung

Das Kommando *aggregates* aktiviert Aggregationsfunktionen wie *Count*, *Max*, *Min*, *Sum*, *Avg* usw.

Solange Sie *aggregates* nicht benutzen, werden bei jeder Schleife über Datensätze im Programm alle Aggregationsfunktionen des ganzen Programms berechnet, gleichgültig, ob die Ergebnisse jemals benötigt werden oder nicht. Dies ist normalerweise kein Problem, bei großen Anwendungen mit sehr vielen Aggregationsfunktionen entstehen dadurch jedoch sehr lange Laufzeiten für *LoopRecs*, *Link*, Unterreports und ähnliche Schleifen.

In diesem Fall rufen Sie vor der Schleife das Kommando *aggregates* auf und geben diejenigen Aggregationsfunktionen an, auf die Sie nach dem Schleifendurchlauf zugreifen wollen. Wenn Sie gar keine Aggregationsfunktionen benötigen, rufen Sie *aggregates* ohne Argumente auf. Dadurch wird die Schleife teilweise bis zu einhundert mal schneller.

Nach einem Aufruf von *aggregates* sind alle Aggregationsfunktionen, die nicht als Argumente angegeben waren, deaktiviert. Deshalb müssen Sie dann vor jeder Schleife, bei der Sie anschließend das Ergebnis einer Aggregationsfunktion auswerten wollen, das Kommando *aggregates* aufrufen.

Beispiel

In diesem Bericht werden all Fahrzeuge und am Schluss die maximale Leistung und der maximale Hubraum ausgegeben. Das aggregates-Kommando bewirkt allerdings, dass nur *Max(Leistung)* korrekt berechnet wird, *Max(Hubraum)* wir als 0 ausgedruckt.

```
.report
.prolog
.primärdatei KFZ
.aggregates Max(Leisung)
.daten
$Leistung, $Hubraum
.epilog
$(Max(Leistung) Max(Hubraum))
```

4.4.3.3 Avg Prozedur

Syntax

```
Avg(Ausdruck): Real
```

Kategorie

Datenbank-Befehl

Erklärung

Bei jeder Datensatzkombination innerhalb eines Reports wird der Ausdruck berechnet, falls die im Ausdruck angesprochenen Tabellen zur Bildung der Satzkombination herangezogen werden. Diese Werte werden addiert. Ebenfalls wird die Anzahl der Kombinationen gespeichert. *Avg* ist nur definiert, wenn wenigstens eine Satzkombination gebildet werden konnte, und liefert dann den Mittelwert *SUM/COUNT*.

Datenbankjob

Der Mittelwert innerhalb von Gruppen kann über den Ausdruck *Avg[i](Ausdruck)* berechnet werden, wobei i für die Nummer der Gruppe steht. *Avg[0]* ist gleichbedeutend mit *Avg* und bezieht sich auf den Datenbereich des Datenbankjobs.

Avg hieß in früheren Version *Mean*.

Bericht

In Berichten muss sowohl die Funktion als Ganzes als auch der Parameter in eckige Klammer gesetzt werden: [AVG([$Feldbezeichner])]

Beispiel

siehe Max.

Siehe auch

Count, Min, Sum, ZCount, ZSum

4.4.3.4 Count Prozedur

Syntax

```
Count(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Anzahl der bearbeiteten Datensatzkombinationen, die auf die angegebene Tabelle zugreifen.

Die Anzahl der Datensatzkombinationen innerhalb von Gruppen kann über den Ausdruck *Count[i](Tabelle)* berechnet werden, wobei i für die Nummer der Gruppe steht. *Count[0]* ist gleichbedeutend mit *Count* und bezieht sich auf den Datenbereich des Berichtes oder Datenbankjobs.

Beispiel (Datenbankjob)

KFZ (=Primärtabelle) ist via Koppelfeld an KUNDEN gebunden.

```
.REPORT
.PROLOGUE
.DEF ItOn=Italic(0)
.DEF ItOff=Italic(1)
.DATA
$(Count(KFZ):3 ") "KFZ.Bezeichnung:30 KFZ.Hersteller)
.SUB KUNDEN
  $(ItOn Count(KUNDEN) ") "KUNDEN(Name, Vorname:30 PLZ:5 Ort) ItOff)
.ENDSUB
.EPILOGUE
Insgesamt sind es $(COUNT(KFZ)) Kraftfahrzeuge und $(SUM(COUNT(KUNDEN)))
Kunden.
```

In einem Modul können Statistikfunktionen in Verbindung mit *Sub*/*EndSub*-Konstruktionen oder mit der Funktion *Link* verwendet werden:

```
PrimFile(KFZ);
Link(KFZ);
Message("Insgesamt sind es " + Str(Count(KFZ)) + "Autos");
```

Siehe auch

Max, Mean, Min, Sum, ZCount, ZSum

4.4.3.5 Max Prozedur

Syntax

```
Max(Ausdruck): Real
```

Kategorie

Datenbank-Befehl

Erklärung

Bei jeder Datensatzkombination innerhalb eines Reports wird der Ausdruck berechnet, falls die im Ausdruck angesprochenen Tabellen zur Bildung der Satzkombination herangezogen werden. Der Wert des Ausdrucks wird aber nur dann gespeichert, wenn das Maximum noch nicht initialisiert oder der neue Wert größer als der bisher gespeicherte ist.

Datenbankjob

Das Maximum innerhalb von Gruppen kann über den Ausdruck *Max[i](Ausdruck)* berechnet werden, wobei i für die Nummer der Gruppe steht. *Max[0]* ist gleichbedeutend mit *Max* und bezieht sich auf den Datenbereich des Berichtes oder Datenbankjobs.

Beispiel

Der folgende Datenbankjob berechnet die statistischen Kenndaten der Tabelle ARTIKEL mit den Feldern "Bestand" und "Einkaufspreis".

```
.REPORT
.PROLOGUE
.PRIMTABLEIS ARTIKEL
.DATA
```

```
$(Bezeichnung:40 Bestand:6 Einkaufspreis:12:2)
.EPILOGUE
Anzahl der verschiedenen Artikel:  $(COUNT(ARTIKEL):10:0)
Gesamtbestand aller Artikel:       $(SUM(Bestand):10:0)
Höchstbestand eines Artikels:      $(MAX(Bestand):10:0)
Kleinster Bestand eines Artikels:  $(MIN(Bestand):10:0)
Durschnittsbestand eines Artikels: $(MEAN(Bestand):10:2)
Höchster Einkaufspreis:            $(MAX(Einkaufspreis):10:2)
Geringster Einkaufspreis:          $(MIN(Einkaufspreis):10:2)
Mittlerer Einkaufspreis:           $(MEAN(Einkaufspreis):10:2)
```

In einem Modul können Statistikfunktionen in Verbindung mit *Sub*/*EndSub*-Konstruktionen oder mit der Funktion *Link* verwendet werden:

```
PrimFile(ARTIKEL);
Link(ARTIKEL);
Message("Höchster Einkaufspreis: " + Str(MAX(Einkaufspreis)));
```

Bericht

In Berichten muss sowohl die Funktion als Ganzes als auch der Parameter in eckige Klammer gesetzt werden: *[MAX([$Feldbezeichner])]*

Siehe auch

Count, Avg, Min, Sum, ZCount, ZSum

4.4.3.6 Min Prozedur

Syntax

```
Min(Ausdruck): Real
```

Kategorie

Datenbank-Befehl

Erklärung

Bei jeder Datensatzkombination innerhalb eines Reports wird der Ausdruck berechnet, falls die im Ausdruck angesprochenen Tabellen zur Bildung der Satzkombination herangezogen werden. Der Wert wird aber nur dann gespeichert, wenn das Minimum entweder noch nicht initialisiert oder der Wert kleiner als das bisherige Minimum ist.

Datenbankjob

Das Minimum innerhalb von Gruppen kann über den Ausdruck *Min[i](Ausdruck)* berechnet werden, wobei i für die Nummer der Gruppe steht. *Min[0]* ist gleichbedeutend mit *Min* und bezieht sich auf den Datenbereich des Berichtes oder Datenbankjobs.

Beispiel

siehe Max

Bericht

In Berichten muss sowohl die Funktion als Ganzes als auch der Parameter in eckige Klammer gesetzt werden: [MIN([$Feldbezeichner])]

Siehe auch

Count, Max, Avg, Sum, ZCount, ZSum

4.4.3.7 Sum Prozedur

Syntax

```
Sum(Ausdruck): Real
```

Kategorie

Datenbank-Befehl

Erklärung

Addiert die Werte des Ausdrucks bei jeder Datensatzkombination, falls die im Ausdruck angesprochenen Tabellen zur Bildung der Satzkombination herangezogen werden.

Datenbankjob

In eckigen Klammern kann der Bereich definiert werden auf den sich die Summenbildung

beziehen soll. *Sum[0](Ausdruck)* ist dabei gleichbedeutend mit *Sum(Ausdruck)* und bezieht sich auf den Datenbereich eines Berichtes oder Datenbankjobs. Die Summe über den Bereich der ersten Gruppe wird mit *Sum[1](Ausdruck)* abgefragt. In Berichten ist es möglich mehrere Gruppenbereiche einzusetzen. Die Indizierung der Summe entspricht hier jeweils der Gruppennummer.

Beispiel

siehe Max

Bericht

In Berichten muss sowohl die Funktion als Ganzes als auch der Parameter in eckige Klammer gesetzt werden: [SUM([$Feldbezeichner])]

Siehe auch

Count, Max, Min, Avg, ZCount, ZSum

4.4.3.8 ZCount Prozedur

Syntax

```
ZCount(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Zählt die Datensatzkombinationen innerhalb eines Reports, falls die angegebene *Tabelle* zur Bildung der Satzkombination herangezogen wird. Die Anzahl bezieht sich auf die aktuelle Druckseite oder, falls im Datenbankjob eine Gruppe definiert wurde, auf die Gruppe.

Hinweis: Verwenden Sie in Berichten anstelle von *ZCount* die Funktion *Count* zum Zählen der Datensatzkombinationen innerhalb einer Gruppe.

Beispiel

```
.REPORT
.PROLOGUE
.PRIMTABLEIS KUNDEN
.SETACCES PLZ.IND
.GP 0
.VAR G_alt=""
.GROUP PLZ[1]

$(ZCOUNT(KUNDEN) " Kunden im PLZ-Bereich " G_alt)

.DATA
$(Name,Vorname:40 PLZ:6 Ort)
.EPILOGUE
```

Siehe auch

Count, Max, Min, Avg, Sum, ZSum

4.4.3.9 ZSum Prozedur

Syntax

```
ZSum(Ausdruck): Real
```

Kategorie

Datenbank-Befehl

Erklärung

Addiert die Werte des Ausdrucks bei jeder Datensatzkombination, falls die im Ausdruck angesprochenen Tabellen zur Bildung der Satzkombination herangezogen werden. Der Wert von *ZSum* bezieht sich auf die aktuelle Druckseite oder, falls im Datenbankjob eine Gruppe definiert wurde, auf die Gruppe.

Hinweis: Verwenden Sie in Berichten anstelle von *ZSum* die Funktion *Sum* zur Bildung von Zwischensummen.

Beispiel

```
.report
.header
.if $Page>1
Übertrag: $(SUM(Gesamtpreis):12:2)
.end
.footer
          ------------
Summe:    $(ZSUM(Gesamtpreis):12:2)
          ============
Gesamt:   $(SUM(Gesamtpreis):12:2)
.data
$(Posten:29 Gesamtpreis:12:2)
.epilogue
```

Siehe auch

Count, Max, Min, Avg, Sum, ZCount

4.4.4 Volltextsuche

Die Funktionen für die Volltextsuche dienen zum Indizieren der Datensätze und zur eigentlichen Suche:

MarkTable	Markiert alle Datensätze, die eine Volltext-Suchbedingung erfüllen
ScanRecs	Indiziert alle Datensätze einer Tabelle neu
ScanRec	Indiziert einen einzelnen Datensatz einer Tabelle neu

4.4.4.1 MarkTable Prozedur

Syntax

```
MarkTable(Table, IndexTable: Integer; SearchExpr: String; ExtABC, Operators:
String; Mask: Integer; RelTable: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion *MarkTable* stellt die Volltextsuche für die Programmierung unter TurboPL zur Verfügung. Voraussetzung ist selbstverständlich die Existenz eines Volltextindex für die betreffende Tabelle. Über die TurboDB Studio-Oberfläche wird die Volltextsuche über den Menüpunkt *Suchen/Nach Stichwörtern...* des Datenfensters gestartet.

Datensätze, die der übergebenen Suchbedingung entsprechen, werden markiert. Die Anzahl der Treffer kann mit der Funktion *NMarks* ermittelt werden.

Table	Ist die Tabelle, für die eine Volltextsuche durchgeführt werden soll.
IndexTable	Ist die Stichworttabelle, die für die Suche herangezogen werden soll.
SearchExpr	Hier wird die Suchbedingung als Zeichenkette übergeben. Gesuchte Feldinhalte können dabei mit den logischen Operatoren UND, ODER und NICHT, symbolisiert durch die Zeichen , + - verknüpft werden. Die Symbole für die Operatoren können mit dem Parameter Operators der Funktion umbelegt werden. Der ODER-Operator hat dabei die niedrigste Priorität, AND und NICHT sind gleichrangig. Es können auch die Joker * und ? in der Suchbedingung erscheinen.
ExtABC	Ist eine Zeichenkette, die Erweiterungen des herkömmlichen Alphabets enthalten kann. Falls keine Erweiterung erforderlich ist, wird einfach ein Leerstring übergeben.
Operators	Hier kann eine Zeichenkette mit den anzuwendenden Zeichen für die logischen Operatoren übergeben werden. Die Vorgaben sind: ',' für UND, '+' für ODER und '-' für NICHT. Soll die Vorgabe erhalten bleiben, hier einfach einen Leerstring übergeben.
Mask	Wird zur Zeit nicht benutzt. Übergeben Sie hier einfach 0.
RelTable	Name der Relationstabelle für die aktuelle Abfrage. Da zwischen Table und

IndexTable theoretisch mehrere Relationen bestehen können muss hier angegeben werden, auf welche sich die Auswertung beziehen soll. Diese Relationstabelle hat den selben Namen wie das betreffende Relationsfeld in Table.

Das Funktionsergebnis ist im Erfolgsfall 0 oder einer der folgenden Fehlernummer:

201	Ungültige Index-Tabelle
202	Ungültige Relation
203	Schließende Klammer erwartet
204	Ungültiges Zeichen

Beispiel

Der folgende Aufruf sucht im Volltextindex, der für die Tabelle LITERATUR in Form der Stichworttabelle VTINDEX erstellt wurde, nach einem Eintrag, der die Begriffe *Anhalter* und *Marvin* enthält, aber nicht den Begriff *ADAC*.

```
ActivateForm("LITERATUR.Formular")
DelMarks(LITERATUR)
MarkTable(LITERATUR, VTINDEX, "Anhalter Marvin -ADAC", "", "", 0, STICHWORT)
IF NMarks(LITERATUR) > 0
      SetSortOrder("Markierung")
END
```

Siehe auch

Volltextindex, ScanRecs, ScanRec

4.4.4.2 ScanRec Prozedur

Syntax

```
ScanRec(MainTable, IndexTable, RelTable: Integer; Fields(Column1 [, Column2,
..ColumnN]); ExtABC: String; MaxFrequency: Integer; ContraIndex: String; Mode:
Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

ScanRec wird verwendet um einen bestehenden Volltextindex für eine Tabelle mit dem aktuellen Datensatz zu aktualisieren.

Die einzelnen Parameter entsprechen dabei denen von *ScanRecs*.

Beispiel

Für die Tabelle LITERATUR wurde mit *ScanRecs* ein Volltextindex angelegt (Vgl. Beispiel zu *ScanRecs*)

Nach einer Neueingabe kann der Volltextindex nun mit dem aktuellen Datensatz aufgefrischt werden

```
ScanRec(LITERATUR, INDEX, STICHWORT, Fields(Kurzbeschreibung, Inhalt), "",
1000, "")
```

Siehe auch

Volltextindex, MarkTable ScanRecs

4.4.4.3 ScanRecs Prozedur

Syntax

```
ScanRecs(MainTable, IndexTable, RelTable: Integer; Fields(Column1 [, Column2,
..ColumnN]); ExtABC: String; MaxFrequency: Integer; ContraIndex: String; Mode:
Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

ScanRecs erzeugt einen Volltextindex für eine Tabelle. Im Normalfall wird man den Index mit Hilfe des in die Oberfläche integrierten Assistenten erstellen (Tabelle/Volltextindex im Menü des

Datenmodells). Für spezielle Fälle kann auf die Funktion *ScanRecs* zurückgegriffen werden.

Die Parameter haben die folgende Bedeutung:

MainTable	Ist die Tabelle, für die der Volltextindex erstellt werden soll.
IndexTable	Ist die Tabelle, in der die Stichwörter abgelegt werden.
RelTable	Ist die Relationstabelle, die die Verknüpfungsinformation von MainTable und IndexTable enthält. Der Name der Tabelle setzt sich aus dem Namens des Relations-Feldes der Tabelle MainTable und der Kennung REL zusammen.
Fields	Als vierter Parameter muss die Funktion Fields übergeben werden. Dieser spezielle Funktion sind als Parameter die Spalten zu übergeben, deren Inhalt bei der Bildung des Volltextindex zu berücksichtigen ist. Über Koppelfeld-Notation können auch Spalten verknüpfter Tabellen einbezogen werden.
ExtABC	Soll das normale Alphabet erweitert werden, kann hier eine Zeichenkette mit zusätzlichen Zeichen übergeben werden. Wird ein Leerstring (") übergeben kommt das herkömmliche Alphabet zum Einsatz.
MaxFrequency	Hier kann eine Obergrenze für die Häufigkeit des Auftretens eines Begriffes angegeben werden. Wird ein Stichwort öfter gefunden, wird es nicht in den Index aufgenommen. Da es meistens wenig hilfreich ist zu wissen, dass ein bestimmtes Wort in beinahe jedem Datensatz zu finden ist, sollte hier in bezug auf die Dateigröße eine sinnvolle Grenze angegeben werden. Dies führt nicht zuletzt bei einer Suche zu einer Steigerung der Performance.
ContraIndex	Hier kann der Name einer Textdatei angegeben werden, die einen Kontraindex enthält. Wörter die im Kontraindex enthalten sind werden nicht in den Volltextindex aufgenommen, was wiederum der Performance zugute kommt. Klassische Wörter für einen Kontraindex sind "der", "die", "das", "ein", "und" u.s.w. Die Textdatei muss die auszuschließenden Begriffe zeilenweise enthalten, pro Zeile ein Begriff. Wird ein Leerstring "" angegeben werden alle Wörter in den Volltextindex aufgenommen.
Mode	Über diesen Parameter kann die Arbeitsweise von ScanRecs eingestellt werden. Übergeben werden Zahlen die jeweils für eine bestimmte Arbeitsweise stehen. Eine Kombination kann durch Addition der Werte erfolgen. 0 dürfte für die meisten Anwendungen empfehlenswert sein. • 0: Aufbau einer neuen Stichworttabelle. Erstellen aller benötigten Indexe (ID, INR) • 1: Verwenden einer bereits bestehenden Stichworttabelle. Mehrere Tabellen können sich durchaus eine Stichworttabelle teilen. Ein Neuaufbau des Index wäre in diesem Fall fatal. • 2: Neue Wörter werden nicht aufgenommen. Es werden nur Verknüpfungen zu bereits in der Stichworttabelle enthaltenen Begriffen erstellt. • 4: Ist reserviert • 8: Ist reserviert

Der Rückgabewert ist die Anzahl der indizierten Stichwörter.

Um einen neuen Volltextindex zu erstellen sind folgende Schritte nötig:

1. Erzeugen Sie eine neue Tabelle. Die erste Spalte der Tabelle muss eine alphanummerische Spalte zur Aufnahme der Stichwörter sein. Die zweite Spalte muss vom Typ Auto-Nummer sein.
2. Öffnen Sie den Designer für die Tabelle, für die der Index erzeugt werden soll. Fügen Sie eine neue Spalte vom Typ Relation in die Tabelle ein und geben Sie als angekoppelte Tabelle die in Schritt 2 neu erzeugte Stichworttabelle an. Beim Restrukturieren der Tabelle wird automatisch die Relationstabelle angelegt.
3. Aufruf von *ScanRecs*.

Um einen bereits bestehenden Volltextindex aufzufrischen, genügt ein erneuter Aufruf von ScanRecs, oder falls nur mit dem aktuellen Datensatz aufgefrischt werden soll *ScanRec*

Beispiele

Der folgende Aufruf von *ScanRecs* erzeugt einen Volltextindex für die Tabelle LITERATUR. Die Tabelle verfügt über ein Relations-Feld Stichwort, über das die Verknüpfung mit der Tabelle INDEX hergestellt wird. Es werden die Spalten Kurzbeschreibung und Inhalt zur Bildung des Volltextindex herangezogen:

```
ScanRecs(LITERATUR, INDEX, STICHWORT, Fields(Kurzbeschreibung, Inhalt), "",
1000, "", 0)
```

Jetzt soll noch der Name des Autors berücksichtigt werden, der sich in einer über das Koppelfeld Autor angekoppelten Tabelle befindet. Zusätzlich wird die maximale Häufigkeit für das Auftreten eines Begriffes auf 200 gesetzt und die Verwendung des Kontraindex festgelegt, der in der Datei Kontra.txt abgelegt ist.

```
ScanRecs(LITERATUR, INDEX, STICHWORT, Fields(Kurzbeschreibung, Inhalt,
Autor.Name), "", 200, "Kontra.txt", 0)
```

Siehe auch

Volltextindex, MarkTable ScanRec

4.4.5 Indexe verwalten

4.4.5.1 Indexe verwalten

Inormationen über Indexe

4.4.5.2 Access Prozedur

Syntax

```
Access(Tabelle: Integer; Zeichenkette: String): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Legt den Zugriff für die angegebene Tabelle fest. Die Zeichenkette darf sein:

- "Markierung"
- "Nummer"
- Der Name eines vorhandenen Indexes der Tabelle.

Liefert die Zugriffsnummer nach Ausführung der Funktion:

-2	Markierung
-1	Nummer
sonst	Indexnummer

Access hat eine ähnliche Aufgabe wie *SetAccess*, ist im Gegensatz zu diesem aber dynamisch. Dadurch kann die Sortierung zur Laufzeit bestimmt werden.

Beispiel

```
Access(KFZ, 'KFZBEZ.IND')
Access(KFZ, 'Markierung')
Access(KFZ, 'Nummer')
```

Siehe auch

IndName, IndNo, SetAccess, Sortierung

4.4.5.3 DelIndex Prozedur

Syntax

```
DelIndex(Tabelle: Integer; IndexNo: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Löscht den angegebenen Index. Dies ist nur möglich, wenn der Index nicht gerade von einer Anwendung benutzt wird. Das Ergebnis ist bei Erfolg 0, sonst eine Fehlernummer.

Siehe auch

IndDef, IndName, IndNo, RegenAll, RegenInd

4.4.5.4 GenIndex Prozedur

Syntax

```
GenIndex(Tabelle: Integer; Indexbeschreibung, IndexName: String [, Eindeutig:
Integer] [, Kapazität: Integer]): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Generiert einen neuen Index für die Tabelle. Die *Indexbeschreibung* ist eine Liste der indizierten Felder. Jedes Feld kann mit einem Minus versehen werden, um die Sortierung umzudrehen. Die Indexbeschreibung kann auch eine Formel sein. Über den optionalen Parameter *Eindeutig* kann die Erstellung eines Unique-Index gesteuert werden. Mögliche Werte:

0	Mehrfacheinträge im Index sind erlaubt (Vorgabe)
1	Indexeinträge müssen eindeutig sein

Mit dem Parameter *Kapazität* wird festgelegt, für wieviele Einträge der Index ausgelegt werden soll. Der Standard-Wert ist 10.000.000, wenn die Tabelle mehr Einträge aufnehmen soll, muss man hier den entsprechenden Wert angeben. Die echte Kapazität des Index kann größer sein als der hier angegebene Wert. Ab TableLevel 4, können Sie diesen Parameter auch für große Tabellen weglassen, weil dann die Kapazität der Tabelle verwendet wird.

Das Ergebnis ist die Index-Nummer des neuen Indexes. Wenn ein Fehler aufgetreten ist, ist das Ergebnis 0 und die Fehlerinformation steht im Error-Objekt.

Beispiel

```
vardef Res: Integer;
Res := GenIndex(KUNDEN, "PLZ,Ort,Straße", "KUNDORT", 0)
if Res = 0
   Message('Fehler ' + Str(Error.Number) + ': ' + Error.Description);
end;
```

Siehe auch

IndDef, IndName, IndNo, RegenAll. RegenInd, DelIndex

4.4.5.5 IndDef Prozedur

Syntax

```
IndDef(Tabelle, Indexnummer [, Optionen: Integer]): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Indexbeschreibung des Index mit der angegebenen *Indexnummer*. Der Parameter *Optionen* ist die Summe dieser Angaben:

1	Angabe der Feldlänge unterdrücken
2	Felder als Feldnummern angeben (ansonsten als Feldnamen)

Die Vorgabe für den Wert *Optionen* ist 0.

Beispiel

```
IndDef(KFZ, 1)          -> "Bezeichnung:30, Modelljahr"
IndDef(KFZ, 1, 1) -> "Bezeichnung, Modelljahr"
IndDef(KFZ, 1, 2) -> "$1:30, $2"
IndDef(KFZ, 1, 3) -> "$1, $2"
```

Siehe auch

GenIndex, IndName, IndNo, RegenAll, RegenInd

4.4.5.6 IndName Prozedur

Syntax

```
IndName(Tabelle, Indexnummer: Integer): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Namen des Index mit der angegebenen *Indexnummer*. Alle vorhandenen Indexe einer Tabelle sind durchnummeriert. Die automatischen Indexe Markierung, Satznummer und Auto-Nummer haben dabei die Nummern -2, -1 und 0. Ab 1 werden die benutzerdefinierten Indexe angesprochen.

Beispiel

```
IndName(KUNDEN, 0)  ->  "KUNDEN.INR"
IndName(KFZ, 2)     ->  "KFZBEZ.IND"
```

Siehe auch

Access, IndNo, SetAccess, Sortierung

4.4.5.7 IndNo Prozedur

Syntax

```
IndNo(Table: Integer): Integer
IndNr(Tabelle: Integer): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Nummer des aktuellen Zugriffs für die angegebene Tabelle. Alle vorhandenen Indexe einer Tabelle sind durchnummeriert. Die automatischen Indexe Markierung, Satznummer und Auto-Nummer (*.INR) haben dabei die Nummern -2, -1 und 0. Ab 1 werden die benutzerdefinierten Indexe angesprochen.

Beispiel

```
IndNo(KFZ)  ->  1
```

Siehe auch

Access, IndName, SetAccess,Sortierung

4.4.5.8 RegenAll Prozedur

Syntax

```
RegenAll(Tabelle: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Regeneriert sämtliche Indexe einer Tabelle. Liefert immer 0.

Beispiel

```
RegenAll(KUNDEN)
```

Siehe auch

GenIndex, IndName, IndNo, RegenInd

4.4.5.9 RegenInd Prozedur

Syntax

```
RegenInd(Tabelle, Indexnummer: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Regeneriert den Index der Tabelle mit der angegebenen Indexnummer. Liefert immer 0. Falls ein Fehler auftritt, wird eine Ausnahme ausgelöst.

Beispiel

```
RegenInd(KUNDEN, 1)
```

Siehe auch

GenIndex, IndName, IndNo, RegenAll

4.4.6 Datensätze

4.4.6.1 Datensätze

Prozeduren, die auf einzelne Datensätze angewendet werden, um die zu lesen, zu schreiben zu sperren etc.

4.4.6.2 append Kommando

Syntax

```
.append Satzbeschreibung
```

Kategorie

Datenbank-Befehl

Erklärung

append fügt einen neuen Datensatz in die Tabelle ein und benötigt dazu natürlich das Recht zur Neueingabe von Datensätzen. Die Syntax ist recht einfach. Dem Kommando wird zunächst der Name der betroffenen Tabelle nachgestellt, dann, in runden Klammern, eine Folge von Feldzuweisungen der Art "Feld = Ausdruck". Die einzelnen Feldzuweisungen werden durch Komma getrennt. Alle Felder, denen kein neuer Wert zugewiesen wird, bleiben unverändert.

Beispiel

Sämtliche offenen Posten werden addiert und das Ergebnis in einen neuen Satz der Tabelle OFFEN eingetragen.

```
.PRIMTABLEIS BESTELL
.VAR Gesamtbetrag =0
.SUB nicht BESTELL.Bezahlt
.VAR Gesamtbetrag=Gesamtbetrag+POSTEN.Menge*ARTIKEL.Einzelpreis
.ENDSUB
.DO APPEND OFFEN(Abschlußdatum=Today,Betrag=Gesamtbetrag)
```

Das Kommand kann auch durch entsprechende ReadRec/WriteRec-Konstruktionen ersetzt werden:

```
APPEND KUNDEN(Name="Fehner", Vorname="Günther")
```

wird dann zu:

```
ReadRec(KUNDEN, 0)
KUNDEN.Name := "Fehner"
KUNDEN.Vorname := "Günther"
WriteRec(KUNDEN, FileSize(KUNDEN)+1)
```

Siehe auch

replace Kommando

4.4.6.3 DelRec

Syntax

```
DelRec(Tabelle [, Satznummer: Integer]): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Löscht den Satz mit der angegebenen physikalischen Satznummer aus der Tabelle. Liefert die physikalische Nummer des folgenden Datensatzes, wobei der aktuelle Zugriff berücksichtigt wird. Wenn die Satznummer nicht angegeben ist, wird der gerade aktuellen Datensatz der Tabelle gelöscht. Wenn ein Fehler auftritt, wird eine Ausnahme ausgelöst.

Beispiel

Löschen des 12-ten Datensatzes:

```
DelRec(KUNDEN, 12)
```

Löschen des aktuellen Datensatzes:

```
DelRec(KUNDEN)
```

Dies ist gleichbedeutend mit

```
DelRec(KUNDEN, RecNo(KUNDEN))
```

Siehe auch

ReadRec, RecNo, WriteRec

4.4.6.4 EditOff

Syntax

```
EditOff(Tabelle: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Eine mit *EditOn* veranlasste Satzsperre wieder aufheben.

Beispiel

Siehe EditOn

4.4.6.5 EditOn

Syntax

```
EditOn(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Versucht eine Satzsperre auf den aktuellen Datensatz der Tabelle einzurichten. Gelingt dies, so liefert die Funktion den Wert 1, andernfalls 0. Falls die Satzsperre eingerichtet werden konnte, muss anschließend die Funktion *EditOff* aufgerufen werden. Damit dies auch bei einem Fehler der Fall ist, muss der Code nach einem *EditOn* immer mit einem *try* - except - *finally* Block gesichert sein.

Beispiel

```
procedure Mutation
    vardef xBreak: Integer
    xBreak := GetPara("NB"); SetPara("NB 0"); .. Kein Abbruch mit ESC möglich
    ShowWait("Satz wir gebucht" + CHR(13) + CHR(10) + "Bitte warten...")
    repeat
       if EditOn(BUCHUNG)
          try
             SetPara("NB 1") .. Abbruch ausschließen
             BUCHUNG.gebucht := JA
             WriteRec(BUCHUNG, RecNo(BUCHUNG))
          except
             Message('Die Buchung konnte nicht korrekt abgeschlossen werden.');
          finally
             EditOff(BUCHUNG)
          end;
          SetPara("NB " + Str(xBreak)) ..Abbruch wie vorher
          HideWait
```

```
         return
      else
         .. In Sperrungsschleifen immer warten, sonst eventuell Blockade!
         Pause(10)
      end
   until 0
endproc
```

4.4.6.6 Ersetzen

Syntax

```
Ersetzen(Bedingung, Ersetzung: String)
ReplaceFields(Condition, Replacement: String)
```

Kategorie

Datenbank-Befehl

Erklärung

Auf alle Datensätze, die die Bedingung erfüllen wird die Ersetzung angewandt. Die Ersetzung enthält Zuweisungen an Datenfelder der Tabelle, die durch Komma getrennt sind.

Entspricht dem Menüpunkt *Suchen/Suchen & Ersetzen* im Menü des Datenfensters.

Beispiel

```
Ersetzen("Produktname = 'Korinus'", "Produktname := 'Corinus', Preis :=
22.22");
```

4.4.6.7 FindRec

Syntax

```
FindRec(Tabelle: Integer; Gesucht, Index: String; Modus: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Bei *Gesucht* handelt es sich um eine Zeichenkette, deren Aufbau der Indexhierarchie entsprechen muss, wobei die einzelnen Hierarchiestufen durch Komma getrennt werden. Die Funktion sucht im angegeben Index diese Zeichenkette und liefert die Datensatzummer des ersten gefunden Datensatzes. Falls mit dem *Modus* 0 gesucht wird, ist die Indexinformation dieses Satzes gleich oder (ordnungsgemäß) größer als Gesucht, im *Modus* 1 ist die Indexinformation exakt gleich Gesucht (mit Ausnahme der Groß/Kleinschreibung).

Das Ergebnis ist 0, wenn kein Datensatz gefunden wurde.

Die Parameter *Index* und *Modus* können entfallen. In diesem Fall werden aktueller Index und Modus 0 verwendet.

Beispiel

```
procedure Suche_nach_Kunden;
   vardef Rec: Integer;
   T-Eingabe := "";
   if Input("Name", "Suche nach Kunden")
      Rec := FindRec(KUNDEN, T-Eingabe, "NAMEN.IND", 1)
      if Rec > 0
         ShowRec(Rec)
      else
         Message(T-Eingabe + " wurde nicht gefunden", "Meldung");
      end;
   end;
endproc
```

Siehe auch

FirstRec, NextRec, ReadRec, Suchen, MitBedingung, WriteRec

4.4.6.8 FirstRec

Syntax

```
FirstRec(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die physikalische Satznummer des ersten Datensatzes der angegebenen Tabelle bezogen auf deren aktuellen Zugriff. Das Ergebnis ist 0, wenn entweder die Tabelle leer ist, oder im Falle des Zugriffs "Markierung" keine Datensätze markiert sind oder bei einem gesetzten Filter keine Datensätze in diesen Filter fallen.

Beispiel

siehe NextRec.

Siehe auch

DelRec, FindRec, FirstRec, NextRec, PrevRec, ReadRec, WriteRec

4.4.6.9 GetField

Syntax

```
GetField(Tabelle, Feldnummer: Integer): String
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert den Inhalt eines Datenfeldes als Zeichenkette. Die Besonderheit dabei ist die große Flexibilität, was den Datenbestand anbelangt. Die Konvertierung des Feldinhaltes erfolgt nach diesen Regeln:

Feldtyp	Ergebnis
Memo, Unicode Memo oder Bild/Klang	"Memo" oder "Blob", wenn das Feld nicht leer ist, ansonsten ein Leerstring.
Kopplung	Autonummer des angekoppelten Datensatzes oder 0 als String.
Relation	
Aufzählung	Text des Aufzählungswertes
Datum, Zeit, Zeitstempel	Datumswert im TurboDB-eigenen Format, z.B. 28.11.2004 14:03:08.365
Fließkommazahl	Zahlenwert im TurboDB-eigenen Format, z.B. 4.56
Zeichenketten (Ansi und Unicode)	im Original zurückgeliefert
Alle übrigen Datentypen	als Zeichenkette zurückgegeben

Beispiel

```
ReadRec(KUNDEN, FindRec(KUNDEN, 'Bolte,Paula', KUNDEN.ID))
GetField(KUNDEN, LabelNo(KUNDEN, 'Name'))    ->  "Bolte"
IF GetField(KUNDEN, LabelNo(KUNDEN, 'Bemerkung')) <> ''
   Message('Im Memo des aktuellen Datensatzes steht was drin')
ELSE
   Message('Das Memo ist leer')
END
```

Siehe auch

SetField

4.4.6.10 GetRec

Syntax

```
GetRec(Tabelle: Real; Satz: Record): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Sichert den kompletten Inhalt eines Datensatzes in einem Satzpuffer. Vorsicht ist bei Memos und Blobs geboten. Es werden nur die im Satzpuffer enthaltenen Verweise auf die Memos/Blobs gesichert, die Inhalte werden nicht berücksichtigt. Daher sollten Memo und Blob-Felder immer gesondert behandelt werden.

Das Ergebnis der Funktion liefert den Fehlerstatus der Operation:

0 alles in Ordnung

< 0 Ein Wert von -X bedeutet, es fehlen X Bytes, um den kompletten Satz zu speichern.

> 0 Positive Werte sind Fehlercodes laut Fehlertabelle.

Beispiel

Im folgenden Beispiels wird ein Satzinhalt gesichert und später wieder zurückgespeichert:

```
procedure SatzSicherung
    vardef SatzSpeicher: record KUNDEN
    GetRec(KUNDEN, SatzSpeicher)
    ..Machen Sie mit KUNDEN, was immer Sie wollen
    PutRec(KUNDEN, Satzspeicher)
endproc
```

Siehe auch

PutRec

4.4.6.11 LastRec

Syntax

```
LastRec(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die physikalische Satznummer des letzten Datensatzes der angegebenen Tabelle bezogen auf deren aktuellen Zugriff. Das Ergebnis ist 0, wenn entweder die Tabelle leer ist oder im Falle des Zugriffs "Markierung" keine Datensätze markiert sind oder bei einem gesetzten Filter keine Datensätze in diesen Filter fallen.

Beispiel

siehe PrevRec.

Siehe auch

FirstRec, NextRec, PrevRec, ReadRec, WriteRec

4.4.6.12 ModifyRec

Syntax

```
ModifyRec(Table: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Bereitet den Satzpuffer der Tabelle für das Editieren vor. Dazu wird der Datensatz für Schreibzugriffe gesperrt. Die Änderung muss mit *PostRec* bestätigt werden. Wenn *ModifyRec* aufgerufen wird bevor die vorhergehende Änderung mit *PostRec* geschrieben wurde, wird die vorhergehende Änderungen zurückgenommen. Falls die Funktion nicht durchgeführt werden kann,

wird ein Laufzeitfehler ausgelöst, sie liefert kein Ergebnis zurück.

Siehe auch

NewRec, PostRec

4.4.6.13 MoveBegin

Syntax

```
MoveBegin(Table: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Stellt den aktuellen Datensatz an den Anfang der Tabelle, noch vor dem ersten Datensatz. Ein darauf folgendes *ReadNext* liest dann den ersten Datensatz. Wenn die Funktion nicht ausgeführt werden kann, wird ein Laufzeitfehler ausgelöst.

Beispiel

Durch das Verhalten von *MoveBegin*, sind Schleifen über die Datensätze sehr elegant und wenig fehleranfällig zu programmieren:

```
MoveBegin(MYTABLE);
while ReadNext(MYTABLE)
    ..Mit dem Datensatz etwas tun
end;
```

Siehe auch

MoveEnd, NewRec, PostRec

4.4.6.14 MoveEnd

Syntax

```
MoveEnd(Table: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Stellt den aktuellen Datensatz an das Ende der Tabelle hinter den letzten Datensatz. Ein darauffolgendes *ReadPrev* liest dann den letzten Datensatz. Wenn die Funktion nicht ausgeführt werden kann, wird ein Laufzeitfehler ausgelöst.

Beispiel

Durch das Verhalten von *MoveEnd*, sind Schleifen über die Datensätze sehr elegant und wenig fehleranfällig zu programmieren:

```
MoveEnd(MYTABLE);
while ReadPrev(MYTABLE)
    ..Mit dem Datensatz etwas tun
end;
```

Siehe auch

MoveBegin, NewRec, PostRec, ReadNext, ReadPrev

4.4.6.15 NextRec

Syntax

```
NextRec(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die physikalische Satznummer des nächsten Datensatzes der angegebenen Tabelle bezogen auf deren aktuellen Zugriff. Das Ergebnis ist 0, wenn kein Datensatz mehr vorhanden ist.

Beispiel

Komplette Bearbeitung einer Tabelle

```
VARDEF RNo : REAL
RNo:=FirstRec(KUNDEN)
WHILE RNo>0
      ReadRec(KUNDEN,RNo)
      ...Hier wird der Kunde bearbeitet
      RNo:=NextRec(KUNDEN)
END
```

Siehe auch

FirstRec, LastRec, PrevRec, ReadRec, WriteRec

4.4.6.16 NewRec

Syntax

```
NewRec(Table: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Bereitet den Satzpuffer von *Table* für einen neuen Datensatz vor. Dabei werden alle Felder mit den korrekten Vorgabewerten gefüllt. Falls die Tabelle eine Auto-Nummer-Spalte besitzt, wird die neue Auto-Nummer bestimmt und in den Satzpuffer eingetragen. Ein mit *NewRec* angelegter Datensatz muss mit *PostRec* in die Tabelle geschrieben werden. Die Funktion löst einen Laufzeitfehler aus, wenn sie nicht ausgeführt werden kann.

Beispiel

```
try
   NewRec(WINEASY);
   WINEASY.Feld1 := DateStr(Today);
   PostRec(WINEASY);
except
   Message("Neuer Eintrag konnte nicht anfgefügt werden:" + Error.Message);
end;
```

Siehe auch

ModifyRec, PostRec

4.4.6.17 PostRec

Syntax

```
PostRec(Table: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Schreibt den aktuellen Datensatz in die Tabelle. Der Datensatz muss zuvor mit *NewRec* oder *ModifyRec* in den Editiermodus geschaltet worden sein.

Die Funktion löst einen Laufzeitfehler aus, wenn sie nicht ausgeführt werden kann.

Beispiel

siehe NewRec

Siehe auch

NewRec, ModifyRec

4.4.6.18 PrevRec

Syntax

```
PrevRec(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die physikalische Satznummer des vorherigen Datensatzes der angegebenen Tabelle bezogen auf deren aktuellen Zugriff. Das Ergebnis ist 0, wenn kein Datensatz mehr vorhanden ist.

Beispiel

Komplette umgekehrte Bearbeitung einer Tabelle

```
VARDEF RNo : REAL
RNo := LastRec(KUNDEN)
WHILE RNo > 0
      ReadRec(KUNDEN,RNo)
      ...
      RNo := PrevRec(KUNDEN)
END
```

Siehe auch

FirstRec, LastRec, NextRec, ReadRec, WriteRec

4.4.6.19 PutRec

Syntax

```
PutRec(Tabelle: Integer; Satz: Record)
```

Kategorie

Datenbank-Befehl

Erklärung

Speichert einen mit *GetRec* gesicherten Datensatz wieder in die Tabelle zurück. Der Datensatz wird dadurch nicht geschrieben, d.h. ohne ein darauf folgendes *WriteRec* macht *PutRec* keinen Sinn.

Beispiel

Siehe GetRec

4.4.6.20 ReadNext

Syntax

```
ReadNext(Tabelle: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Liest den auf den aktuellen Datensatz folgenden Datensatz in den Satzpuffer der Tabelle. Liefert 1, wenn noch ein Datensatz zum Lesen vorhanden war und 0 falls nicht. Löst im Fehlerfall einen Laufzeitfehler aus.

Siehe auch

ReadPrev

4.4.6.21 ReadPrev

Syntax

```
ReadPrev(Table: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Lädt den dem aktuellen Datensatz vorhergehenden Datensatz in den Satzpuffer. Liefert 1, wenn noch ein Datensatz zum Lesen vorhanden war und 0 falls nicht. Falls die Funktion nicht durchgeführt werden kann, wird ein Laufzeitfehler ausgelöst.

Siehe auch

ReadNext, MoveEnd

4.4.6.22 ReadRec

Syntax

```
ReadRec(Table, RecordNo: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Liest den Datensatz mit der angegebenen physikalischen Satznummer. Liefert die Satznummer oder 0, wenn der Satz nicht gelesen werden konnte.

Achtung

Falls als Satznummer 0 angegeben wird, wird ein leerer Datensatz zur Verfügung gestellt, der mit *WriteRec* in die Tabelle geschrieben werden kann. Diese Funktionalität dient hauptsächlich der Rückwärtskompatibilität und sollte nicht mehr eingesetzt werden. Verwenden Sie stattdessen die Funktionen *NewRec* und *PostRec*.

Beispiel

```
try
   ReadRec(WINEASY, 12212);
except
   Message("Fehler beim Lesen aus Tabelle WINEASY:" + Error.Message);
end;
```

Siehe auch

FindRec, FirstRec, LastRec, NextRec, PrevRec, ShowRec, WriteRec

4.4.6.23 replace Kommando

Syntax

```
.replace Satzbeschreibung
```

Kategorie

Datenbank-Befehl

Erklärung

Im aktuellen Datensatz der Tabelle werden die Ersetzungen aus der Satzbeschreibung vorgenommen und der Datensatz dann in die Tabelle geschrieben. Die in der Satzbeschreibung nicht erwähnten Datenfelder behalten dabei ihren vorherigen Wert. Die Satzbeschreibung hat die Form

```
TABELLE(Feld1 = NeuerWert1, Feld2 = NeuerWert2, ...)
```

Falls *replace* in einem Subreport auf Tabellen angewandt wird, die selbst zur Bearbeitung des Subreports herangezogen werden, darf durch keine der Feldzuweisungen der aktuelle Zugriff beeinflußt werden. Es ist hier in vielen Fällen günstiger, wenn in einem ersten Subreport die betroffenen Datensätze markiert werden, und die eigentlichen Ersetzungen in einem zweiten Subreport vollzogen werden, in dem der Zugriff auf Markierung gesetzt wird.

Beispiel

In der Tabelle ARTIKEL werden alle Einzelpreise um 10% erhöht.

```
.primTableIs ARTIKEL
.setAccess Nummer
.sub
.do replace ARTIKEL(Einzelpreis=Einzelpreis*1.10)
.endsub
```

Das *replace*-Kommando kann man auch durch normale Datenbankprozeduren ersetzen,

allerdings ist dann erheblich mehr zu schreiben. So entspricht zum Beispiel das Kommando

```
REPLACE KUNDEN(LetzterKontakt=Today)
```

der Prozedursequenz

```
KUNDEN.LetzterKontakt := Today
WriteRec(KUNDEN, RecNo(KUNDEN))
```

Siehe auch

append Kommando

4.4.6.24 SetField

Syntax

```
SetField(Tabelle, Feldnummer: Integer; Zeichenkette: String)
```

Kategorie

Datenbank-Befehl

Erklärung

Überträgt den Inhalt der Zeichenkette in das Feld mit der angegebenen Feldnummer. Die Änderung betrifft den aktuellen Satz der betroffenen Tabelle. Die Änderung wird aber erst durch *WriteRec* in die Datenbank-Tabelle übertragen. Das korrekte Format für den Wert ist bei *GetField* beschrieben.

Beispiel

Der vierte Datensatz der KUNDEN-Tabelle wird verändert:

```
var NeuerWert = 'Müller'
ReadRec(KUNDEN, 4)
SetField(KUNDEN, LabelNr(KUNDEN, 'Name'), NeuerWert)
WriteRec(KUNDEN, RecNo(KUNDEN))
```

Siehe auch

GetField

4.4.6.25 SetRecord

Syntax

```
SetRecord(Ausgangstabelle, Zieltabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Diese Funktion überträgt den aktuellen Datensatz der Ausgangstabelle in den aktuellen Satz der Zieltabelle. Dabei findet kein reiner Kopiervorgang statt, sondern die Felder werden den Feldbezeichnern entsprechend zugeordnet wie z.B. auch beim Import. Auch die laufende Nummer der Ausgangstabelle wird direkt übernommen, wenn die entsprechenden Felder den selben Namen haben.

Nach dem Aufruf von *SetRecord* muss der Satz der Zieltabelle noch per *WriteRec* geschrieben werden. Falls der Datensatz an die Zieltabelle nicht angehängt wird, muss eine sinnvolle laufende Nummer eingetragen werden.

Beispiel

```
vardef Rec: Integer;
PrimFile(RECHNUNG);
Link(RECHNUNG.Tag < 1.1.1997, SetMark(RECHNUNG, RecNo(RECHNUNG)));
Access(RECHNUNG, "Markierung");
Rec := FirstRec(RECHNUNG)
while ReadRec(RECHNUNG, Rec)
    ReadRec(RECHBAK, 0);
    SetRecord(RECHNUNG, RECHBAK);
    WriteRec(RECHBAK, FileSize(RECHBAK)+1);
    Rec := NextRec(RECHNUNG);
end
```

4.4.6.26 WriteRec

Syntax

```
WriteRec(Tabelle, Satznummer: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Schreibt den aktuellen Satz an die physikalische Position der angegebenen Satznummer. Falls es sich bei der Satznummer um einen Wert innerhalb der bisherigen Tabellengröße (FileSize) handelt, wird der dort befindliche Satz überschrieben. Ist die Satznummer (um 1) größer als die bisherige Tabellengröße, so wird der Satz angehängt und die Tabelle entsprechend erweitert.

Der Rückgabewert ist die Datensatz-Nummer. Wenn ein Fehler auftritt, wird 0 zurückgegeben und ein Laufzeitfehler ausgelöst.

Beispiel

Das folgende Makro erzeugt einen neuen Datensatz und hängt ihn an die Tabelle an, nachdem zuvor eine Reihe von Vorbelegungen stattgefunden haben.

```
procedure NeuerSatz
    ReadRec(TableNo, 0)    ..  neuen Satz erzeugen
    SetField(TableNo, 1, "Vorbelegung")
    SetField(TableNo, 2, "Vorbelegung")
    ...
    WriteRec(TableNo, 1+FileSize(TableNo))
endproc
```

Siehe auch

FirstRec, LastRec, NextRec, PrevRec, ReadRec, SetField

4.4.7 Memos und Blobs

4.4.7.1 Memos und Blobs

Funktionen zum Lesen und Schreiben von Memos und Blobs.

ReadMemo	Memo einlesen
CopyMemo	Inhalt eines Memos in Datei schreiben
MemoLen	Länge eines Memos ermitteln
DelMemo	Memo löschen
EmbedBlob	Blob in Tabelle kopieren
LinkBlob	Blob mit Tabelle verknüpfen
BlobSize	Größe eines Blobs ermitteln
GetLinkedFile	Dateiname eines verknüpften Blobs ermitteln
CopyBlob	Inhalt eines Blobs in Datei schreiben
ClearBlob	Blob löschen

4.4.7.2 BlobSize Prozedur

Syntax

```
BlobSize(Blob: Feld): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion gibt die Größe des übergebenen BLOB-Feldes in Bytes zurück. Beim übergebenen BLOB muss es sich um ein eingebettetes Bild, bzw. um einen eingebetteten Klang handeln. Um die Dateigröße eines verknüpften BLOBs zu ermitteln verwenden Sie bitte die Funktionen *GetLinkedFile* und *FirstDir*.

Beispiel

Die Prozedur liefert die Größe des BLOB-Feldes Sample, unabhängig davon ob es sich um ein verknüpftes oder ein eingebettetes BLOB handelt:

```
PROCEDURE GetBlobSize: Integer;
   VarDef BlobPath: String;
   BlobPath := GetLinkedFile(Sample);
   IF Length(BlobPath) = 0
      Return BlobSize(Sample);
   ELSE
      VarDef s: String;
      s := FirstDir(BlobPath, "");
      Return Val(s[15, 10]);
   END;
ENDPROC;
```

Siehe auch

GetLinkedFile, EmbedBlob, LinkBlob

4.4.7.3 CopyMemo Prozedur

Syntax

```
CopyMemo(Field: TableField; FileName: String; CharSet: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Bei dem angegeben Feld muss es sich um ein Memofeld handeln. Der Inhalt dieses Memofeldes wird in eine Textdatei kopiert, deren Name durch die Zeichenkette Datei bestimmt wird. Besteht die Textdatei bereits, wird diese überschrieben. Ansonsten wird eine neue Textdatei erzeugt. Der dritte Parameter ist optional.

CharSet	1	Datei wird im DOS-Zeichensatz (OEM) angelegt
	0	Datei wird im Windows-Zeichensatz (ANSI) angelegt (Vorgabe)

Der Rückgabewerte ist bei Erfolg 0, sonst kommt die Fehlernummer.

Beispiel

Alle Memos einer Datei werden in getrennte Textdateien geschrieben. Die Namen der Textdateien sind "PARTx", wobei x der physikalische Satznummer entspricht.

```
LoopRecs(KUNDEN, CopyMemo(Bemerkung, "PART" + Str(RecNo(KUNDEN))))
```

Siehe auch

ReadMemo, MemoLen, MemoStr

4.4.7.4 ClearBlob Prozedur

Syntax

```
ClearBlob(Field: BlobField)
```

Kategorie

Datenbank-Befehl

Erklärung

Der Inhalt es Blobs im aktuellen Datensatz der Tabelle wird gelöscht. Die Änderung mit mit dem nächsten *PostRec* oder *WriteRec* in die Tabelle geschrieben. Falls das angegebene Feld kein Blob-Feld ist, wird ein Fehler ausgelöst.

Beispiel

Dieses Code-Fragment löscht das Foto aus dem ersten Datensatz der Tabelle.

```
ReadRec(FOTOS, 1);
ModifyRec(FOTOS);
ClearBlob(Foto);
PostRec(FOTOS);
```

Siehe auch

ReadMemo, MemoLen, MemoStr

4.4.7.5 CopyBlob Prozedur

Syntax

```
CopyBlob(Field: BlobField; FileName: String): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Bei dem angegeben Feld muss es sich um ein Blob-Feld handeln. Der Inhalt dieses Blob-Felds wird in eine Datei gespeichert, deren Name durch die Zeichenkette *FileName* bestimmt wird. Besteht die Datei bereits, wird sie überschrieben. Ansonsten wird eine neue Datei erzeugt.

Der Rückgabewerte ist bei Erfolg 0, sonst kommt die Fehlernummer.

Siehe auch

LinkBlob, EmbedBlob

4.4.7.6 DelMemo Prozedur

Syntax

```
DelMemo(Active: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

DelMemo(0) schaltet das Überschreiben von gelöschten Memos durch neu hinzukommende aus. Dadurch werden die freigegebenen Blöcke nicht überschrieben, was zu Test-Zwecken und zur Diagnostik manchmal ganz nützlich sein kann. Der Platz für die gelöschten Memos wird aber verschwendet, so dass man die Option für den regulären Betrieb mit *DelMemo(1)* wieder aktivieren sollte.

4.4.7.7 EmbedBlob Prozedur

Syntax

```
EmbedBlob(Feld: TableField; Quelldatei: String; Format: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Lädt die unter *Quelldatei* angegebene Bild- oder Klangdatei in das Datenfeld. Natürlich muss das Feld ein Blob-Feld sein. *Format* ist optional und definiert den Typ des Bildes. Die möglichen Werte sind unter *LinkBlob* beschrieben. *EmbedBlob* liefert bei Erfolg die Größe der eingebetteten Datei zurück, ansonsten löst sie einen Laufzeitfehler aus.

Beispiel

Ein Makro-Button eines Formulares soll die Prozedur *ImportiereGrafikdatei* ausführen

```
procedure ImportiereGrafikdatei
    T-Eingabe := "";
    if ChoosePicture("Wählen Sie ein Bild aus", 127)
       EmbedBlob(BASE.Sample, T-Eingabe);
       WriteRec(BASE, RecNo(BASE))
    end
endproc
```

Wie bei allen Zuweisungen an Tabellenfelder wird auch von *EmbedBlob* lediglich der aktuelle Datenpuffer beschrieben. Damit der Datensatz tatsächlich geschrieben wird, ist ein anschließender *WriteRec* notwendig.

Siehe auch

PlaySound, LinkBlob, ChoosePicture, ClearBlob, CopyBlob

4.4.7.8 GetLinkedFile Prozedur

Syntax

```
GetLinkedFile(Blob: Feld): String
```

Kategorie

Datenbank-Befehl

Erklärung

Die Funktion liefert den Pfad zum übergebenen Blob-Feld. Voraussetzung dafür ist natürlich, dass es sich um eine verknüpfte Bild- bzw. Klangdatei handelt. Für eingebettete Blobs wird eine leere Zeichenkette zurückgegeben. In diesem Fall verwenden Sie die Funktion *PlaySound* zum Abspielen des Klanges.

Beispiel

Die Prozedur spielt einen vorhandenen Sound. Ist das Blob leer und der Editier- bzw. Neueingabemodus aktiviert, wird ein Dateidialog zum Verknüpfen einer WAVE-Datei angezeigt.

```
PROCEDURE Hörbeispiel;
      IF Sample
           VarDef BlobPfad: String;
           BlobPfad := GetLinkedFile(Sample);
           IF Length(BlobPfad) > 0
                PlayMedia(GetLinkedFile(Sample));
           ELSE
                PlaySound(Sample);
           END;
      ELSE
           IF GetMode <> 0
                T-Eingabe := '*.WAV';
                IF ChooseFile('Sound laden')
                        IF Upper(RightStr(T-Eingabe, 4)) = '.WAV'
                                     LinkBlob(Sample, T-Eingabe, 3);
                        ELSE
                                     Message( 'Ungültges Dateiformat',
'Fehlerhafte Eingabe');
                        END;
                END;
           END;
      END;
ENDPROC;
```

Siehe auch

BlobSize, MediumPause, MediumSpielen, MediumStop, PlaySound

4.4.7.9 LinkBlob Prozedur

Syntax

```
LinkBlob(Feld: TableField; Quelldatei: String; Format: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Ersetzt den aktuellen Inhalt des angegebenen Datenfeldes durch eine Verknüpfung mit der unter Quelldatei angegeben Datei. Natürlich muss das Feld ein Blob-Feld sein. Die Funktion liefert 0 oder einen Fehlercode zurück.

Der Parameter *Format* ist optional und legt den Typ des Bildes fest.

0	Der Typ wird aus der Dateiendung bestimmt, wenn möglich
1	Bitmap, d.h. BMP- oder DIB-Datei
2	Paintbrush-Bild, d.h. PCX-Datei
3	Klang, d.h. WAV-Datei

4 Metadatei, d.h. Dateiendung WMF
5 CompuServe-Format, d.h. Dateiendung GIF
6 JPEG, d.h. Dateiendung JPG

Achtung

Anders als andere Funktionen, die einen Datensatz verändern, schreibt *LinkBlob* den Datensatz sofort in die Tabelle.

Beispiel

Der folgende Code verknüpft die ausgewählte wave-Datei mit dem Blob-Feld im ersten Datensatz. Ein *anschließendes WriteRec* ist nicht nötig.

```
ReadRec(MUSIC, 1);
T-Eingabe := "*.wav";
if ChooseFile("Wave-Datei laden") and Upper(RightStr(T-Eingabe, 4)) = ".WAV)
   LinkBlob(Sample, T-Eingabe, 3);
end;
```

Siehe auch

BlobSize, EmbedBlob, GetLinkedFile, PlaySound, ClearBlob, CopyBlob

4.4.7.10 MemoLen Prozedur

Syntax

```
MemoLen(Memo: Field): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

MemoLen ermittelt die Anzahl der Zeichen des übergebenen Memos.

Beispiel

```
VarDef ml: Real
ml := MemoLen(KUNDEN.Bemerkung)
Message("Im Feld 'Bemerkung' sind " + Str(ml) + " Zeichen enthalten")
```

Siehe auch

CopyMemo, ReadMemo, MemoStr

4.4.7.11 ReadMemo Prozedur

Syntax

```
ReadMemo(Feld: TableField; FileName: String; Mode: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Mit dieser Funktion wird eine externe ASCII-Datei in ein Memofeld gelesen. *FileName* bestimmt den Namen der Textdatei. Der Mode gibt an, ob das Memo komplett überschrieben wird, oder ob der Text an den bisherigen Inhalt angefügt wird.

Mode = 0 -> Text wird angehängt

Mode = 1 -> Memo wird ersetzt

Diese Funktion hat als ersten Parameter ein Memofeld. Wird etwas anderes angegeben, so erhalten Sie die Fehlermeldung "Illegaler Typ". Die Funktion *ReadMemo* ist die Umkehrung von *CopyMemo* und ermöglicht das Einlesen einer externen ASCII-Datei in ein bestehendes Memofeld. Deshalb muss als zweiter Parameter der Name einer existierenden Textdatei angegeben werden.

Als dritter Parameter kann wiederum eine Zahl angegeben werden. 1 bedeutet hier, dass der Memoinhalt durch die Textdatei überschrieben wird. 0 führt dazu, dass die Textdatei an den bisherigen Inhalt angehängt wird.

Auch hier wird sowohl die Memodatei aktualisiert, als auch der zugehörige Datensatz mit der neuen Memonummer versehen und in die Datei zurückgeschrieben.

ReadMemo liefert im Erfolgsfall 0, sonst einen Fehlercode.

Beispiel

```
ReadMemo(KUNDEN.Bemerkung, "A:BERICHT.TXT", 1)
```

Siehe auch

CopyMemo, MemoStr, MemoLen

4.4.8 SQL-Befehle

4.4.8.1 SQL-Befehle

SQL (Standard Query Language) ist eine standardisierte Abfragesprache für Datenbanken. Die Verwendung von SQL ist in *TurboDB Studio* nicht nötig aber möglich. Vorteile ergeben sich, wenn

- Das Ergebnis aus einer komplizierten Tabellen-Kombination zwischengespeichert werden soll, ohne eine Datei auf der Festplatte anzulegen, oder
- Auf eine Kombination aus Tabellen mit *ReadRec* und ähnlichen Funktionen zugegriffen werden soll, oder
- Eine Kompatibilität mit anderen Datenbankwerkzeugen hergestellt werden soll.

In diesem Abschnitt werden die Befehle der Makrosprache beschrieben, die für die Arbeit mit SQL-Kommandos notwendig sind. Eine detaillierte Behandlung des unterstützten SQL-Dialekts mit dem Namen *TurboSQL* finden Sie in dem entsprechenden Kapitel.

Englischer Name	Name	Kurzbeschreibung
ExecSQL	SQLAusführen	führt ein SQL-Kommando ohne Ergebnismenge aus
OpenSQL	ÖffneSQL	führt ein SQL-Kommando mit Ergebnismenge aus

4.4.8.2 ExecSQL Prozedur

Syntax

```
ExecSQL(Command: String): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Führt ein SQL-Kommando ohne Ergebnismenge aus, z.B. ein INSERT, DELETE, UPDATE, CREATE, ALTER oder DROP. Der Rückgabewert ist die Anzahl der betroffenen Datensätze. Wenn während der Ausführung des SQL-Kommandos ein Fehler auftritt, wird -1 zurückgegeben und ein Laufzeitfehler ausgelöst.

Beispiel

```
if ExecSQL("INSERT INTO TESTDATA (Column1) VALUES('Otto')") < 0
    Message("Fehler in SQL-Befehl: " + Error.Description);
end;
```

Siehe auch

OpenSQL

4.4.8.3 OpenSQL Prozedur

Syntax

```
OpenSQL(Command: String): Tabelle
```

Kategorie

Datenbank-Befehl

Erklärung

Führt ein SQL-Kommando mit Ergebnismenge aus, z.B. ein SELECT. OpenSQL funktioniert analog zu OpenDB mit einem SQL-Kommando statt mit einem Tabelle-Namen.

Beispiel: Hier wird eine SQL-Abfrage gestartet und die Ergebnismenge anschließend durchlaufen.

```
vardef RltnHdl: Integer;
RltnHdl := OpenSQL('SELECT * FROM TESTDATA');
vardef Rec: Integer;
Rec := FirstRec(RltnHdl);
while Rec > 0
      ReadRec(RltnHdl, Rec);
   Message(GetField(RltnHdl, 1));
   Rec := NextRec(RltnHdl);
end
CloseDB(RltnHdl);
```

Siehe auch

ExecSQL

4.4.9 Interne Markierungen

4.4.9.1 Interne Markierungen

Prozeduren zum Setzen und Lesen von Markierungen.

4.4.9.2 AndMarks Prozedur

Syntax

```
AndMarks(Tabelle: Integer; Markierungen: Integer[]): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Nach dieser Operation sind nur noch diejenigen Datensätze der Tabelle markiert, die vorher markiert waren und deren Satznummern in dem Array enthalten sind. Zurückgegeben wird 0 oder ein Fehlercode.

Beispiel

```
..zweiten und fünften Eintrag der Kunden-Tabelle markieren
SetMark(KUNDEN, 2)
SetMark(KUNDEN, 5)
..Markierungen merken
VarDef MarkList: Real[100]
GetMarks(KUNDEN, MarkList)
..aktive Markierungen löschen
DelMarks(KUNDEN)
..den ersten und fünften Datensatz markieren
SetMark(KUNDEN, 1)
SetMark(KUNDEN, 5)
..logische Verknüpfung der aktiven und gemerkten Markierungen durchführen
AndMarks(KUNDEN, MarkList)
..Jetzt ist nur noch der fünfte Datensatz markiert
```

Siehe auch

DelMark, DelMarks, GetMarks, IsMark, NMarks, NotMarks, PutMarks, SetMark, SortMark

4.4.9.3 DelMark Prozedur

Syntax

```
DelMark(Tabelle, Satznummer: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Entfernt die Markierung des Satzes mit der angegeben Satznummer aus der Tabelle. Falls der Satz nicht markiert war, hat die Funktion keine Wirkung. Als Ergebnis wird die Satznummer zurückgegeben.

Beispiele

Löscht die Markierung des 207-ten Datensatzes:

```
DelMark(KFZ,207)
```

Löscht die Markierung des gefundenen Datensatzes:

```
if Input("Kunde", "Löschen")
    DelMark(KUNDEN, FindRec(KUNDEN, T-Eingabe, "KUNDEN.ID", 1))
end
```

Siehe auch

AndMarks, DelMarks, GetMarks, IsMark, NMarks, NotMarks, PutMarks, SetMark, SortMark

4.4.9.4 DelMarks Prozedur

Syntax

```
DelMarks(Tabelle: Integer)
```

Erklärung

Löscht sämtliche Markierungen der Tabelle. Ergebnis ist immer 1.

Beispiel

```
DelMarks(POSTEN)
```

Siehe auch

AndMarks, DelMark, GetMarks, IsMark, NMarks, NotMarks, PutMarks, SetMark, SortMark

4.4.9.5 GetMarks Prozedur

Syntax

```
GetMarks(Tabelle: Integer; var Markierungen: Integer[])
```

Kategorie

Datenbank-Befehl

Erklärung

Speichert die Satznummern der aktuellen internen Markierungen in einem Integer-Array. Interne Markierungen entstehen durch Kennzeichnen mit einem Sternchen an der Oberfläche und anschließendem *Ansicht/Nur markiert Datensätze* bzw. durch die Tabellen-Funktion *SetMark*.

Erster gültiger Index des Arrays ist 0, letzter ist *NMarks(Tabelle) - 1*.

Falls das Array zu klein ist, um alle Markierungen aufzunehmen, wird die Größe entsprechend angepasst, dabei wird ein mehrdimensionales Array gegebenenfalls zu einen eindimensionalen Array gemacht.

Beispiel

Das folgende Makro löscht alle markierten Datensätze:

```
procedure MarkierteLöschen
    vardef Markierungen: Integer[0]
    vardef i, n: Integer
    if n := NMarks(FileNo)
       GetMarks(FileNo, Markierungen)
       NLoop(i, n - 1, DelRec(FileNo, Markierungen[i]))
    end
endproc
```

Siehe auch

AndMarks, DelMark, DelMarks, IsMark, NMarks, NotMarks, PutMarks, SetMark, SortMark

4.4.9.6 IsMark Prozedur

Syntax

```
IsMark(Tabelle, Satznummer: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Prüft, ob der Datensatz mit der übergebenen physikalischen Satznummer markiert ist (Ergebnis 1) oder nicht (Ergebnis 0).

Beispiel

Die Markierung für den aktuellen Datensatz wird umgeschaltet:

```
IF IsMark(KUNDEN, RecNo(KUNDEN))
      DelMark(KUNDEN, RecNo(KUNDEN))
ELSE
      SetMark(KUNDEN, RecNo(KUNDEN))
END
```

Siehe auch

AndMarks, DelMark, DelMarks, GetMarks, NMarks, NotMarks, PutMarks, SetMark, SortMark

4.4.9.7 MarkRel Prozedur

Syntax

```
MarkRel(Table, RelTable: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

MarkRel markiert in einer über ein Relationsfeld mit *Table* verknüpften Tabelle alle Datensätze (interne Markierungen), die mit dem aktuellen Datensatz von *Table* verknüpft sind. Diese Aufgabe kann man ohne weiteres mit *LoopRecs*/*Link* implementieren. Bei großen Tabellen und vielen verknüpften Datensätzen ist *MarkRel* jedoch wesentlich schneller, erfordert aber, dass die Relations-Indexe zuvor durch einmaligen Aufruf der Funktion *RelIndex* vorbereitet werden.

Der Name der Relationstabelle *RelTable* ergibt sich aus dem Namen des Relationsfeldes und der Dateiendung rel. Die Zieltabelle, in der markiert wird, ist eindeutig durch die Relationstabelle festgelegt.

Mehr Informationen über die Funktionsweise von *RelIndex* und *MarkRel* erhalten Sie unter *RelIndex*. Sie sollten auch diesen Abschnitt unbedingt lesen, weil hier auch die Einschränkungen von *RelIndex*/*MarkRel* erläutert sind.

Beispiel

Siehe RelIndex

Siehe auch

RelIndex, Relationsfeld

4.4.9.8 NMarks Prozedur

Syntax

```
NMarks(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Anzahl der Markierungen einer Tabelle.

Beispiel

```
Link(KUNDEN.Name = "Meier", SetMark(KUNDEN, RecNo(KUNDEN)));
IF NMarks(KUNDEN)>0
      Access(KUNDEN,"Markierung");
      Attach;
ELSE
      Message("Keine Datensätze gefunden","Hinweis",0)
END
```

Siehe auch

AndMarks, DelMark, DelMarks, GetMarks, IsMark, NotMarks, PutMarks, SetMark, SortMark

4.4.9.9 NotMarks Prozedur

Syntax

```
NotMarks(Tabelle: Integer; Markierungen: Integer[]): Integer;
```

Kategorie

Datenbank-Befehl

Erklärung

Führt zwischen den gespeicherten und den aktuellen Markierungen eine UND NICHT-Verknüpfung durch. Nach dieser Operation sind in der Tabelle alle diejenigen Datensätze markiert, die im Array Markierungen eingetragen sind und in der Tabelle vorher nicht markiert waren. Der Rückgabewert ist 0 oder ein Fehlercode.

Siehe auch

AndMarks, DelMark, DelMarks, GetMarks, IsMark, NMarks, PutMarks, SetMark, SortMark

4.4.9.10 PutMarks Prozedur

Syntax

```
PutMarks(Tabelle: Integer; Markierungen: Integer[])
```

Kategorie

Datenbank-Befehl

Erklärung

Markiert alle Datensätze mit denjenigen Nummern, die durch das Array übergeben wurden. Diese Funktion bildet das Gegenstück zu *GetMarks*.

Beispiel

Das folgende Beispiel führt eine UND-Suche auf eine Tabelle durch, d.h. es werden nur diejenigen Datensätze markiert, die bereits markiert sind und auf die eine Bedingung zutrifft.

```
procedure MarkiereUnd(Bedingung: String)
    vardef Markierungen: Integer[8000]
    vardef i: Integer
    setAccess Markierung
    i := 0
    sub _Bedingung
       Markierungen[i] := RecNo(FileNo)
       i := i + 1
    endsub
    DelMarks(FileNo)
    PutMarks(FileNo, Markierungen)
endproc
```

Siehe auch

AndMarks, DelMark, DelMarks, GetMarks, IsMark, NMarks, NotMarks, SetMark, SortMark

4.4.9.11 RelIndex Prozedur

Syntax

```
RelIndex(RelTable, IndexNo: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Mit dieser Funktion wird ein Index einer Relationstabelle (Dateikennung REL) für den schnellen Zugriff mittels *MarkRel* vorbereitet. Übergeben wird die Relationstabelle und die Nummer des Indexdatei: 1 für IN1 sowie 2 für IN2. Damit ermöglicht der Aufruf mit 1 das Markieren in der Eltern-Tabelle (also der Tabelle, auf die das Relationsfeld zeigt), während der Aufruf mit 2 das Markieren in der Kind-Tabelle (Tabelle mit dem Relationsfeld) erlaubt.

Hintergrund und Warnung

Normalerweise enthalten die Indexe in1 und in2 einer Relationstabelle eine Sortierung der Datensätze der Relationstabelle nach den Auto-Nummern der angekoppelten Datensätze. in1

bezieht sich dabei auf die Kind-Tabelle und in2 auf die Eltern-Tabelle. Durch die Funktion *RelIndex* werden zusätzlich die physikalischen Satznummern der jeweils anderen Tabelle in den Index eingetragen. Also die Satznummern der Eltern-Tabelle in den in1 und die Satznummern der Kind-Tabelle in den in2. Auf diese Weise kann *MarkRel* die über das Relationsfeld verknüpften Datensätze finden, ohne die Relationstabelle lesen zu müssen. Deshalb ist es bei sehr vielen Datensätzen in der Relationstabelle erheblich schneller.

Die von *RelIndex* eingefügten Informationen können jedoch nicht automatisch gewartet werden. Der Einsatz von *RelIndex* und *MarkRel* ist also in erster Linie für statische Daten interessant. Für andere Zwecke wird der Standard-Weg über *LoopRecs*/*Link* in Verbindung mit *SetMark* empfohlen.

Beispiel

Die Tabelle LITERATUR verfügt über ein Relationsfeld *Stichwort*, das die Verknüpfung zur Tabelle INDEX herstellt. Ein Datenfenster der Tabelle LITERATUR ist geöffnet. Das folgende Beispiel markiert alle Stichwörter zum aktuellen Literaturhinweis und zeigt sie in einem Formular an.

```
vardef Marks: Integer[0]
MarkRel(LITERATUR, STICHWORT)
if NMarks(INDEX) > 0
   GetMarks(INDEX, Marks)
   OpenForm('INDEX.Formular_INDEX')
   PutMarks(INDEX, Marks)
   Attach
else
   Message("Zum aktuellen Eintrag sind keine Stichwörter vorhanden"
end
```

Damit dies funktioniert muss zuvor einmalig die Relationstabelle mit dem folgenden Aufruf vorbereitet werden:

```
RelIndex(STICHWORT, 1)
```

Die Vorbereitung in der anderen Richtung ist nötig, wenn man mit *MarkRel* zu einem Stichwort in der Tabelle Index alle passenden Literaturstellen finden möchte:

```
RelIndex(STICHWORT, 2)
```

Siehe auch

MarkRel

4.4.9.12 SortMark Prozedur

Syntax

```
SortMark(Tabelle: Integer; Indexbeschreibung: String)
```

Kategorie

Datenbank-Befehl

Erklärung

Sortiert die markierten Datensätze einer Tabelle, wobei als Sortierkriterium die übergebene Indexbeschreibung verwendet wird.

Beispiel

Sortiert alle Datensätze der Tabelle KUNDEN nach den Feldern "PLZ" und "Ort".

```
VARDEF i: REAL
.. alle Sätze markieren
nLoop(i, FileSize(KUNDEN)-1, SetMark(KUNDEN,i+1))
SortMark(KUNDEN, "PLZ,Ort")
```

Siehe auch

AndMarks, DelMark, DelMarks, GetMarks, IsMark, NMarks, NotMarks, PutMarks, SetMark

4.4.10 Netzwerk

4.4.10.1 Netzwerk

Mit den Netzwerk-Funktionen können Sie Sperren auf Tabellen und Datensätze setzen sowie die Benutzer einer Tabelle auflisten lassen. Außerdem sind auch die Funktionen für Transaktionen in dieser Gruppe enthalten.

4.4.10.2 Lock Prozedur

Syntax

```
Lock(Tabelle, ForWrite, DontWait: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Hier handelt es sich um eine Netzwerkfunktion, die in der Einplatzversion immer den Wert 1 liefert.Die Funktion *Lock* dient dem Setzen der elementaren Schreib- beziehungsweise Totalsperre. *Lock* sollte immer dann benutzt werden, wenn Zugriffe auf Datenbanktabellen über mehrere einzelne Funktionen erfolgen, die ihrerseits wiederum Sperren setzen und wieder aufheben (z.B. *WriteRec*). Durch *Lock* ist es nun möglich, zu Beginn der komplexen Datenbankzugriffe einmal die erforderliche Sperre einzurichten, die Funktionen abzuarbeiten und die Sperre mit *Unlock* wieder aufzuheben. Dies bringt im Netz nicht nur eine erhöhte Sicherheit, sondern auch eine gesteigerte Performanz.

Für die Art der Sperre (Parameter ForWrite) gilt:
0 : Schreibsperre
1 : Totalsperre

Für den Parameter DontWait gilt:
0 : Sperre wird unbedingt eingerichtet; Programm wartet
1 : Sperre wird eingerichtet, wenn möglich; Programm wartet nicht

Rückgabewert:
1 : Sperre ist eingerichtet
0 : Sperre ist nicht eingerichtet

Hinweis: Jede erfolgreich eingerichtete Sperre muss mit *Unlock* wieder aufgehoben werden.

Beispiel

Das folgende Beispiel zeigt bereits den Sinn von *Lock*. Alle in diesem Beispiel verwendeten Tabellenzugriffe setzen ihrerseits jeweils vor Abarbeitung die nötigen Sperren und heben diese sofort wieder auf, wenn die Funktion abgearbeitet wurde. Wird allerdings mit *Lock* eine Sperre zu Anfang auf die Datei gelegt, so setzen die einzelnen Funktionen ihrerseits keine Sperren mehr. Die Performanz kann so deutlich verbessert werden. Zudem kommt speziell in diesem Beispiel noch die Tatsache hinzu, dass unter Umständen ein gesicherter Datensatz wieder gelesen werden muss. Im ungünstigsten Fall wird zwischen dem Sichern der Satznummer in *nSaveRec* und dem Zurücksetzen des Satzzeigers mit *ReadRec(KUNDE.DAT, nSaveRec)* der Datensatz von einer anderen Station entfernt.

Somit wäre ein sicheres Lesen des alten Datensatzes gefährdet. Daher sollten solche Vorgänge in einer netzwerkfähigen Entwicklung immer von *Lock* und *Unlock* umgeben sein.

```
PROCEDURE FindName(cSuch: String): Real;
VARDEF nSaveRec, nret : REAL;
      Lock(KUNDE.DAT, 0);
      nSaveRec := RecNo(KUNDE.DAT);
      ReadRec(KUNDE.DAT, FINDREC(KUNDE.DAT, cSuch));
      IF NOT $KUNDE.Nachname LIKE cSuch
           ReadRec(KUNDE.DAT, nSaveRec);
      ELSE
           nRet := 1;
      END
      UnLock(KUNDE.DAT, 0);
      Return nRet
ENDPROC;
```

Siehe auch

EditOff, EditOn, Rollback, TransOff, TransOn, UnLock

4.4.10.3 NetId Prozedur

Syntax

```
NetId: String
```

Kategorie

Datenbank-Befehl

Erklärung

Nur Netzwerkversion.

Gibt den Namen des Computers im Netzwerk zurück. Sie können den Namen des Rechners über den Menüpunkt *Datei/Einstellungen* festlegen.

Beispiel

```
Message("Der Namen dieses Rechners lautet: " + NetId)
```

Siehe auch

NetUsers

4.4.10.4 NetUsers Prozedur

Syntax

```
NetUsers(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Liefert die Anzahl aller Arbeitsplätze, die momentan mit der angegebenen Tabelle arbeiten.

Beispiel

In diesem Beispiel wird auf einfache Weise ermittelt, ob eine Anwendung nur für die Anzahl bezahlter Lizenzen im Netz benutzt wird.

```
procedure CheckUserCount;
   if NetUsers(Kunden) > 5
      Message("Das Programm wurde nur für 5 Arbeitsplätze lizensiert",
"Meldung");
   end;
endproc;
```

Siehe auch

PrivDir, UserNo

4.4.10.5 RollBack Prozedur

Syntax

```
RollBack(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Führt in der Mehrplatzversion den Zustand der Tabelle wieder herbei, wie er vor dem Aufruf von *TransOn* war. Liefert 0 (=erfolgreich) oder Fehlercode. In der Einplatzversion liefert die Funktion immer der Wert 0.

Beispiel

siehe TransOn

Siehe auch

EditOff, EditOn, Lock, TransOff, TransOn, Unlock

4.4.10.6 TransOff Prozedur

Syntax

```
TransOff(Tabelle: Integer): Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Beendet eine Transaktion. Gibt 0 zurück falls die Aktion erfolgreich beendet wurde, andernfalls den Fehlercode.

Ist nur in der Mehrplatzversion aktiv. Entspricht der Funktion *Commit* in anderen Datenbanksystemen. Näheres siehe Benutzer-Handbuch.

Beispiel

siehe TransOn

Siehe auch

EditOff, EditOn, Lock, RollBack, TransOn, Unlock

4.4.10.7 TransOn Prozedur

Syntax

```
TransOn(Tabelle: Real): Real
```

Kategorie

Datenbank-Befehl

Erklärung

TransOn legt zunächst eine Totalsperre auf die Tabelle und richtet anschließend ein Protokoll ein, in dem sämtliche Änderungen an der Tabelle gespeichert werden. Aufgrund dieses Protokolls können alle Änderungen bei Bedarf rückgängig gemacht werden. Wird die Funktion *TransOff* ausgeführt, so wird die Totalsperre aufgehoben und das Protokoll gelöscht. Wird hingegen die Funktion *RollBack* ausgeführt, so werden sämtliche Änderungen, die seit dem Aufruf von *TransOn* vorgenommen wurden, wieder rückgängig gemacht.

Die Funktion liefert 0, wenn die Operation erfolgreich ausgeführt werden konnte, andernfalls wird der Fehlerstatus zurückgeliefert.

In der Einplatzversion liefert die Funktion immer 0.

Beispiel

```
IF TransOn(ARTIKEL)=0
      x := FindRec(ARTIKEL,"12345","ARTIKEL.ID",l)
      ReadRec(ARTIKEL, x);
      ARTIKEL.Preis:=ARTIKEL.Preis*1.05;
      WriteRec(ARTIKEL, x);
      TransOff(ARTIKEL);
END
```

Siehe auch

EditOff, EditOn, Lock, TransOff, Rollback, Unlock

4.4.10.8 Unlock Prozedur

Syntax

```
Unlock(Tabelle: Integer, Sperre: Integer)
```

Kategorie

Datenbank-Befehl

Erklärung

Hebt eine mit *Lock* gesetzte Sperre wieder auf.

Beispiel

Siehe Lock

Siehe auch

EditOff, EditOn, Lock, Rollback, TransOff, TransOn

4.4.10.9 UserNo Prozedur

Syntax

```
UserNo: Integer
```

Kategorie

Datenbank-Befehl

Erklärung

Nummer des aktuellen Users, d.h. die Nummer der Arbeitsstation, wie sie bei den Netzwerk-Einstellungen eingetragen wurde.

Beispiel

```
Message("Die Nummer des Arbeitsplatzes ist: " + Str(UserNo));
```

Siehe auch

NetUsers, PrivDir

4.5 Datenbankjobs

4.5.1 Datenbankjobs

Eine Einführung in das Arbeiten mit Datenbankjobs finden Sie unter "Datenbankjobs erstellen".

Ausgabeformate

Ein Ausgabeformat ist ein Ausdruck, der definiert was in welcher Weise gedruckt wird. Im einfachsten Fall ist das ein Text, der eins-zu-eins ausgegeben wird, die nächste Stufe ist ein Verweis auf eine Tabellen-Spalte, deren Inhalt gedruckt wird und dann geht es weiter mit Formeln und formatierten Ausdrücken. All diese Möglichkeiten finden Sie in der Referenz der " Ausgabeformate".

Bereiche

Der Bereich, in dem sich ein Ausgabeformat befindet bestimmt, wann es ausgegeben wird. Alle zur Verfügung stehenden Bereiche sind unter "Bereichskommandos" beschrieben.

Befehle

Eine ganze Reihe von TurboPL-Befehlen sind speziellen für Datenbankjobs zugeschnitten. Sie finden Sie unter "Befehle in Datenbankbankjobs". Darüberhinaus können alle Funktion und sonstigen Befehle in Datenbankjobs ebenfalls benutzt werden.

Maßangaben

Als Maßeinheit für alle Angaben ist per Vorgabe eine durchschnittliche Zeichengröße eingestellt. Bei Größenangaben in horizontaler Richtung ist das eine durchschnittliche Zeichenbreite, bei Größenangaben in vertikaler Richtung eine durchschnittliche Zeichenhöhe. Das bedeutet, dass zum Beispiel in der Ausgabe *$(Wort:4)* eine Proportionalschrift vorausgesetzt, auch der Text "iiiii" Platz hat, der Text "MMMM" aber wohl nicht. Die Maßeinheit kann mit dem Steuerbefehl *MM* geändert werden.

4.5.2 Ausgabeformate

4.5.2.1 Ausgabeformate

Ausgabeformate werden in Datenbankjobs und teilweise auch in Berichten benutzt, um eine textuelle Ausgabe zu formatieren.Das Ausgabeformat wird durch ein $-Zeichen eingeleitet und muss geklammert werden, wenn es mehr als eine einfache Komponente enthält Z.B:

```
.report
```

```
.daten
$(Name:40 Vorname:20 Alter:8)
```

Ein Ausgabeformat besteht im Prinzip aus der Angabe der Namen der auszugebenden Tabellenspalten. Zusätzlich können Sie wörtlichen Text in doppelten Anführungszeichen einfügen, Spaltenbreiten mit Doppelpunkt definieren und Zeilenumschaltungen mit einem Schrägstrich veranlassen. Hier ein paar Beispiele.

Format	**Erläuterung**
`Name Vorname`	Name und Vorname werden hintereinander ausgegeben.
`Name Vorname /`	Nach dem Vornamen wird ein zusätzlicher Zeilenwechsel eingefügt.
`Name:30 Vorname:10`	Feldbreiten für Name und Vorname. Dadurch stehen alle Vornamen sauber untereinander.
`"Name: " Name`	Ausgabe des Namens mit Beschriftung

Der durch Feldbreiten-Angaben entstehende Freiraum kann mit den Zeichen . oder _ aufgefüllt werden, die man direkt an die Feldbreitenangabe anschließt.

```
Name:30. Vorname:10_
```

Das Ausgabeformate kann auch die folgenden Systemvariablen beinhalten:

`heute`	Aktuelles Datum als Zeichenkette
`jetzt`	Aktuelle Uhrzeit als Zeichenkette
`Zeile`	Aktuelle Zeilennummer als Zeichenkette
`Seite`	Aktuelle Seitennummer als Zeichenkette

Mit dem +-Zeichen werden mehrere Komponenten zu einer verbunden. Nachfolgende Formatierungen gelten dementsprechend für die zusammengebundene Komponente. Als Beispiel dient die Ausgabe einer Zeilennummerierung in einer Spalte mit vier Zeichen;

```
Zeile:4 ". " Name + ", " + Vorname
```

In Datenbankjobs kann im Ausgabeformat auch eine zuvor mit *font* definierte Schriftart gesetzt werden. Die Syntax dazu ist:

```
@<FontName>
```

Als Beispiel die Ausgabe in zwei Schriftarten

```
.report
.font Std=Arial,10
.font Hd1=Arial,20,b
.daten
$(@Std Name/ @Hd1 Vorname)
```

Weitere Informationen hierzu finden Sie unter *"font"*.

Darüberhinaus gibt es noch eine ganze Reihe von weiteren Formatierungsmöglichkeiten, die in den folgenden Abschnitten erläutert werden.

4.5.2.2 Zeilentrennung

Zum Trennen von Zeilen können Sie die folgenden Zeichen benutzen:

/ Unbedingter Zeilenwechsel

| Zeilenwechsel, wenn in dieser Ausgabeformat-Komponente schon etwas ausgegeben wurde. Falls nicht, wird kein Zeilenwechsel durchgeführt.

\ Wie |. Wenn der Zeilenwechsel aber nicht durchgeführt wurde, wird er am Ende des aktuellen Datensatzes eingefügt. Dadurch stehen alle leeren Zeilen am Ende, und die Datensätze nehmen alle gleich viel Platz ein.

< Zeilenwechsel vor Ausgabe der Komponenten, falls Komponente nicht leer ist.

4.5.2.3 Formatieren von Zeichenketten

Für Zeichenketten gibt es die folgenden Formatierungen:

Format	**Beispiel**	**Erklärung**
`String`	`Nachname`	unformatiert

`String:n`	`Nachname:30`	linksbündige Ausgabe in einem Feld der Länge n
`String:n:m`	`Nachname:30:3`	linksbündig in einem Feld der Länge n in m Zeilen
`String:n:m:e`	`Nachname:30:3:2`	linksbündig in einem Feld der Länge n in m Zeilen mit e Spalten Einzug ab der zweiten Zeile
`String R_Form n`	`Nachname R_Form 30`	rechtsbündig in einem Feld der Länge n
`String R n`	`Nachname R 30`	wie R_Form
`String C_Form n`	`Nachname C_Form 30`	zentriert in einem Feld der Länge n
`String L_Form n`	`Nachname L_Form 30`	linksbündig in einem Feld der Länge n (längere Strings werden abgeschnitten)
`String L n`	`Nachname L 30`	wie L_Form

Anmerkung

Eine mehrzeilige Formatierung zum Beispiel mit *R_Form 30:3* ist derzeit **nicht** möglich.

4.5.2.4 Formatieren von Zahlen

Für das Formatieren von Zahlen haben Sie die folgenden Möglichkeiten:

Format	Beispiel	Erklärung
`Zahl`	`Betrag`	unformatiert
`Zahl:n`	`Betrag:20`	rechtsbündig in einem Feld der Länge n
`Zahl:n:m`	`Betrag:20:5`	rechtsbündig in einem Feld der Länge n mit m Nachkommastellen
`Zahl L_Form n`	`Betrag L_Form 20`	linksbündig in einem Feld der Länge n
`Zahl L n`	`Betrag L 20`	wie L_Form

Anmerkung

Eine Formatierung von Nachkommastellen über *R_Form* oder linksbündig mit Nachkommastellen mit *L_Form* (zum Beispiel *Zahl L_Form 20:3*) ist derzeit nicht möglich.

4.5.2.5 Memos formatieren

Beim Formatieren von Memos gelten ähnliche Regeln wie bei der Formatierung von Zeichenketten. Wenn das Memo innerhalb einer Spaltenbreite neu umgebrochen wird, so werden im Memo enthaltene Leerzeilen als Absatzgrenze interpretiert:

Format	Beispiel	Erklärung
`Memo`	`Inhalt`	Komplettausgabe über die Breite der Seite
`Memo:n`	`Inhalt`	Komplettausgabe in einem Feld der Breite n
`Memo:256`	`Inhalt:256`	Komplettausgabe, wobei Zeilenumbrüche unterdrückt werden, nützlich für Export in Layout-Systeme.
`Memo:0`	`Inhalt:0`	Komplettausgabe ohne Zeilenumbruch
`Memo:n:m`	`Inhalt:50:5`	Ausgabe von m Zeilen in einem Feld der Breite n
`Memo:0:m`	`Inhalt:0:5`	Ausgabe von m Zeilen ohne Umbruch
`Memo:n:m:e`	`Inhalt:50:5:2`	Ausgabe von m Zeilen in einem Feld der Breite n mit Einzug e ab der zweiten Zeile. Der Einzug wird vom linken Seitenrand aus gemessen.

Anmerkung

Im Datenbankjob muss die Formatierung in runde Klammern eingeschlossen werden: *$(Memo:40:10:10)*

4.5.2.6 Bedingte Ausgabe

Die Ausgabe einer Komponente kann nach bestimmten Regeln unterdrückt werden:

Format	**Erklärung**
`xyz?`	Falls xyz nur aus Leerzeichen besteht, erfolgt keine Ausgabe.
`xyz??(Selektion)`	Falls die Selektion nicht zutrifft, wird die Ausgabe von xyz unterdrückt.
`xyz!`	Die Komponente vor einem ! wird nur dann ausgegeben, wenn sie sich von der entsprechenden Komponente des vorherigen Satzes unterscheidet oder wenn eine neue Seite begonnen wurde. Diese Form der bedingten Ausgabe darf nur einmal pro Ausgabeformat verwendet werden.

Beispiele

Das Ausgabeformat *Nachname:30! Vorname* erzeugt folgende Liste, wenn es zwei Personen mit dem Namen *Huber* gibt.

Dürman	Anton
Huber	Egon
	Erwin
Maier	Franz

4.5.2.7 Berechnungen

Mit der Funktion *Calc* kann jeder beliebige Ausdruck berechnet und im Ausgabeformat verwendet werden.

Beispiel

```
"Preis: " Calc(Anzahl*Einzelpreis)
```

Außerdem stehen die folgenden statistischen Funktionen zur Verfügung:

Funktion	**Erklärung**
Count	Anzahl der Datensätze bis zur aktuellen Position
Count(<Ausdruck>)	Anzahl der Datensätze bis zur aktuellen Position, für die Ausdruck einen Wert ungleich 0 ergibt
Sum(<Ausdruck>)	Summe der Werte des Ausdrucks über alle Datensätze bis zur aktuellen Position
Min(<Ausdruck>)	Minimum der Werte des Ausdrucks über alle Datensätze bis zur aktuellen Position
Max(<Ausdruck>)	Maximum der Werte des Ausdrucks über alle Datensätze bis zur aktuellen Position
Avg(<Ausdruck>)	Mittelwert der Werte des Ausdrucks über alle Datensätze bis zur aktuellen Position
Sum [<Gruppe>](<Ausdruck>)	Summe der Werte des Ausdrucks über alle Datensätze der angegebenen Gruppe bis zur aktuellen Position
Min [<Gruppe>](<Ausdruck>)	Summe der Werte des Ausdrucks über alle Datensätze der angegebenen Gruppe bis zur aktuellen Position
Max [<Gruppe>](<Ausdruck>)	Summe der Werte des Ausdrucks über alle Datensätze der angegebenen Gruppe
Avg [<Gruppe>](<Ausdruck>)	Summe der Werte des Ausdrucks über alle Datensätze der angegebenen Gruppe bis zur aktuellen Position
ZCount(<Ausdruck>)	Anzahl der Datensätze in der Gruppe oder in der aktuellen Seite bis zur aktuellen Position, für die Ausdruck einen Wert ungleich 0 ergibt
ZSum(<Ausdruck>)	Summe der Werte des Ausdrucks über alle Datensätze der Gruppe oder der aktuellen Seite bis zur aktuellen Position

4.5.2.8 Sub Funktion

Syntax

```
Sub(Ausgabeformat)
```

Erklärung

Die hier beschriebene Funktion *Sub* ist nicht identisch mit dem Kommando *sub*, auch wenn die Funktion ähnlich ist. Die Funktion *Sub* arbeitet alle verknüpften Datensätze in einem einzigen Aufruf ab, während das Kommando *sub* einen Bereich in der Prozedur oder im Datenbankjob definiert, der für alle verknüpften Datensätze bearbeitet wird.

Diese Funktion ist nur innerhalb eines Ausgabeformats erlaubt und führt einen eingebetteten Unterreport aus. *TurboDB Studio* analysiert hierzu das Ausgabeformat und bestimmt daraus diejenigen Tabellen, auf die der Unterreport zugreift. Im Anschluss daran werden sämtliche zur aktuellen Primärtabelle gehörenden Datensatzkombinationen mit diesen Tabellen gebildet und für jede davon das Ausgabeformat ausgegeben. Innerhalb des Ausgabeformats gibt es folgende Sonderregelungen:

- Ist die erste Komponente eine *SEL*-Funktion, so wird nicht deren Wert ausgegeben, sondern der Rest des Ausgabeformats wird nur für solche Datensatzkombinationen ausgegeben, die diese Selektion erfüllen.
- Beginnt das eigentliche Ausgabeformat mit einem Komma und besteht das Ausgabeformat aus einer einzigen (beliebig zusammengebundenen) Ausgabekomponente, so werden die den Datensatzkombinationen entsprechenden Ausgaben durch Komma getrennt hintereinander ausgegeben. Auf die letzte Ausgabe folgt kein Komma mehr.
- Beginnt das eigentliche Ausgabeformat mit einem Schrägstrich (=Zeilenumbruch), so werden die einzelnen Ausgaben durch einen Zeilenumbruch ausgegeben. nach der letzten Ausgabe erfolgt kein Zeilenumbruch.
- Während der Unterreport bearbeitet wird, wird der linke Rand vorübergehend auf die Startposition des Unterreports gesetzt.

Beispiele

Im ersten Beispiel werden die Fahrzeuge des Kunden durch Komma getrennt in einem fortlaufenden Text ausgegeben:

```
$(KUNDEN(Name, Vorname):20. SUB(,KFZ.Bezeichnung)/)
Müller, Hubert......Fiat 500, Suzuki Swift, VW Golf
Rastlos, Rudi.......Mercedes 190 SL, Mercedes 300 SL,
                    Jaguar E
Primel, Johanna.....VW Polo, Opel Omega
...
```

Im zweiten Beispiel erfolgt ein Zeilenumbruch nach jedem Fahrzeug:

```
$(KUNDEN(Name, Vorname):20. SUB(/KFZ.Bezeichnung)/)
Müller, Hubert......Fiat 500
                    Suzuki Swift
                    VW Golf
Rastlos, Rudi.......Mercedes 190 SL
                    Mercedes 300 SL,
                    Jaguar E
Primel, Johanna.....VW Polo
                    Opel Omega
...
```

Innerhalb des Subreports wird die Primärtabelle entsprechend umgestellt. Aus diesem Grund kann die Funktion auch verschachtelt eingesetzt werden. Die Funktion liefert kein Ergebnis, mit dem weitergerechnet werden kann.

Siehe auch

Link, Exists

4.5.3 Bereichskommandos

4.5.3.1 Bereichskommandos

Die Bereichskommandos geben den Anfang eines bestimmten Textbereiches an. Das Ende dieses Bereichs wird durch das folgende Bereichskommando definiert. Es können alle Bereichskommandos entfallen. Ist kein Bereich definiert, so wird der Textbereich als Prolog interpretiert, wenn innerhalb des Textes kein Tabellenzugriff erfolgt, andernfalls als Datenbereich.

Es gibt zwei Arten von Datenbankjobs: Reports und Serienbriefe. Sie sind völlig gleich aufgebaut und unterscheiden sich nur dadurch, dass bei Serienbriefen automatisch nach jedem Datensatz ein Seitenvorschub erfolgt. Durch ein Bereichskommando in der ersten Zeile geben Sie an, welche Art von Datenbankjob gemeint ist. Darüber hinaus werden auch Textdateien, die Prozeduren enthalten mit einem solchen Bereichskommando gekennzeichnet.

report	Der folgenden Datenbankjob ist ein Report.
letter/serienbrief	Der folgende Datenbankjob ist ein Serienbrief.

Datenbankjobs werden in verschiedene Bereiche eingeteilt, deren Reihenfolge vorgegeben ist. Dies geschieht durch das entsprechende Kommando, welches durch einen Punkt in der ersten Spalte gekennzeichnet wird. Bereiche können auch ausgelassen werden. Der Bereich zwischen dem Beginn des Datenbankjobs (also dem Befehl report oder letter) und dem ersten Bereichskommando (meistens prolog) wird Initialisierungsbereich genannt.

prologue/prolog	Der erste Kopfbereich, Voreinstellungen für den Datenbankjob, Gesamtüberschrift
header/kopf	Ausgabe einer oder mehrerer Kopfzeilen auf jeder Seite.
footer/fuß	Ausgabe einer oder mehrerer Fußzeilen auf jeder Seite.
groupHeader/gruppenKopf	Für Gruppen-Überschriften
groupFooter/gruppenFuß	Für Gruppen-Zusammenfassungen
data/daten	Ausgabeformat der eigentlichen Datensätze
epilogue/epilog	Wird nach dem letzten Datensatz bearbeitet.

Siehe auch

Datenbankjobs

4.5.3.2 report Kommando

Syntax

```
report
```

Kategorie

Datenbankjobs

Erklärung

Definiert das Modul als Datenbankjob. Dadurch werden Ausgabeformate ausgedruckt und es dürfen die speziellen Befehle für Datenbankjobs eingesetzt werden.

Siehe auch

Datenbankjobs erstellen, letter

4.5.3.3 letter Kommando

Syntax

```
letter
```

Kategorie

Datenbankjobs

Erklärung

Definiert das Modul als Datenbankjob für Serienbriefe. Dadurch wird nach jedem ausgegebenen

Datensatz ein Seitenvorschub eingefügt. Falls Sie zu Beginn eines Datenbankjobs nichts angeben, ist *letter* bzw. *serienbrief* die Voreinstellung.

Siehe auch

Datenbankjobs erstellen, report

4.5.3.4 prologue/prolog Kommando

Syntax

```
prologue
prolog
```

Kategorie

Datenbankjobs

Erklärung

Der Prolog wird einmal am Anfang des Datenbankjobs bearbeitet. Mit dem Steuerkommando *HE* kann festgelegt werden, ob der erste Kopfbereich schon vor dem Prolog ausgegeben wird oder nicht.

Folgende Kommandos dürfen nur im Prolog verwendet werden:

```
primtableis
setaccess
selection
filter
relation
```

Wird ein Subreport im Prolog ausgegeben, so wird die Primärtabelle komplett bearbeitet. Ob diese Bearbeitung zur Berücksichtigung eines eventuell vorhandenen Gruppenbereiches führt, kann mit dem Steuerkommando *GC* bestimmt werden.

Der Textbereich bis zum Beginn des Prologs darf nur Steuerkommandos enthalten. Sie werden vor der Bearbeitung des Datenbankjobs ausgeführt und enthalten die globalen Voreinstellungen.

Enthält der Prolog Tabellenzugriffe, so wird dabei auf den aktuellen Datensatz der Primärtabelle zugegriffen. Der Zugriff auf andere Tabellen als die Primärtabelle liefert hier zufällige Ergebnisse.

Beispiel

Siehe data

Siehe auch

data, epilogue, footer, group, header, Bereichskommandos

4.5.3.5 header/kopf Kommando

Syntax

```
header
```

Kategorie

Datenbankjobs

Erklärung

Der Kopfbereich wird am Anfang jeder Seite bearbeitet. Mit dem Steuerkommando *HE* kann festgelegt werden, ob der erste Kopfbereich schon vor dem Prolog ausgegeben wird oder nicht.

Beispiel

```
.report
.prolog
.let Linie=xMal("-",75)
.header
$Linie
$("Bericht vom "+$heute:30 "Seite "+$Seite r 35)
$Linie
.data
```

Siehe auch

data, epilogue, footer, group, prologue, Bereichskommandos

4.5.3.6 footer/fuß Kommando

Syntax

```
footer
```

Kategorie

Datenbankjobs

Erklärung

Der Fußbereich wird am Ende jeder Seite bearbeitet. Da seine Länge zur Bestimmung der für Daten nutzbaren Zeilenzahl herangezogen wird, muss seine Länge konstant bleiben und darf keine if-Kommandos enthalten. Sowohl im Kopf wie im Fußbereich kann auf die Statistikfunktionen *ZSum* und *ZCount* zurückgegriffen werden, die sich bei fehlendem Gruppenbereich immer auf die Druckseite beziehen.

Beispiel

Siehe data

Siehe auch

data, epilogue, group, header, prologue, Bereichskommandos

4.5.3.7 group/gruppe Kommando

Syntax

```
group <Gruppenausdruck>
gruppe <Gruppenausdruck>
groupHeader <Gruppenausdruck>
gruppenKopf <Gruppenausdruck>
```

Kategorie

Datenbankjobs

Erklärung

Dem Kommando *group* wird als Argument ein beliebiger Ausdruck nachgestellt. Dieses Argument wird der Kurzfunktion mit dem Namen *G_Neu* zugewiesen. Bei jeder Satzkombination des Datenbereiches wird dieser Ausdruck berechnet und mit dem Wert des vorherigen Durchgangs verglichen. Wird eine Änderung festgestellt, so wird der Gruppenbereich bearbeitet. Stellt der Anwender eine Variable mit dem Namen *G_alt* zur Verfügung, so kopiert *TurboDB Studio* den letzten Wert von *G_neu* in diese Variable, bevor diese neu berechnet wird.

Falls ein Gruppenbereich definiert wurde, beziehen sich die Funktionen *ZCount* und *ZSum* auf diese Gruppe.

Damit eine sinnvolle Gruppenbildung vorgenommen werden kann, muss die Ausgabe der Daten sortiert erfolgen, und die Sortierreihenfolge muss der Gruppendefinition partiell entsprechen.

Das Bereichskommand *groupheader* bzw. *gruppenkopf* ist gleichbedeutend mit *group* bzw. *gruppe* . In Kombination mit *groupfooter* bzw. *gruppenfuß* ist es jedoch besser verständlich.

Anmerkung

Aus Kompatibilitätsgründen wird das Steuerkommando *GP* noch unterstützt. Sie sollten aber stattdessen lieber groupHeader und groupFooter verwenden, um Gruppenbereiche vor und nach dem Datenbereich ausgeben zu können.

Beispiel

Stichwortverzeichnis:

```
.report
.prologue
.primtableis LEXIKON
.setaccess LEXIKON.ID
.group LEXIKON.Stichwort[1]

$(Bold(1) G_Neu Bold(0))

.data
$(LEXIKON.Stichwort:30. SUB(,VERWEIS.Seitennummer))
.epilogue
```

Siehe auch

gruppenfuß, G_Neu, G_alt, Bereichskommandos

4.5.3.8 groupFooter/gruppenFuß Kommando

Syntax

```
groupfooter
gruppenfuß
```

Kategorie

Datenbankjobs

Erklärung

Die Kombination *groupHeader* plus *groupFooter* ist dazu gedacht, in einem gruppierten Datenbankjob sowohl vor als auch hinter jeder Gruppe etwas auszugeben. Bei der Verwendung von *group* alleine, kann man ja (mittels Steuerkommando *GP*) nur entweder vor der Gruppe oder nach der Gruppe Ausgaben vornehmen. Die Reihenfolge der Bereiche ist dann wie folgt:

```
.groupHeader
..Hier die Ausgaben zu Beginn jeder Gruppe
.groupFooter
..Hier die Ausgaben am Ende jeder Gruppe
.data
..Hier die Daten innerhalb der Gruppe
.groupHeader
..Hier die Ausgaben zu Beginn jeder Gruppe
.groupFooter
..Hier die Ausgaben am Ende jeder Gruppe
.data
..Hier die Daten innerhalb der Gruppe
```

Siehe auch

group, Bereichskommandos

4.5.3.9 data/daten Kommando

Syntax

```
data
daten
```

Kategorie

Datenbankjobs

Erklärung

Der Datenbereich wird für jede Datensatzkombination bearbeitet. *TurboDB Studio* analysiert, auf welche Tabellen innerhalb des Datenbereiches zugegriffen wird. Zusätzlich werden noch die im Selektions- und Gruppenkommando verwendeten Tabellen herangezogen. Über diese Tabellen werden die relationalen Datensatzkombinationen gebildet.

Beispiel

```
.report
.prologue
.primtableis KUNDEN
.header
Telefonliste vom $Heute
.footer
$Seite
.data
$(Name,Vorname:40. Telefon)
.epilogue
```

Siehe auch

epilogue, footer, group, header, prologue, Bereichskommandos

4.5.3.10 epilogue/epilog Kommando

Syntax

```
epilogue
epilog
```

Kategorie

Datenbankjobs

Erklärung

Der Epilog wird am Ende des Datenbankjobs (aber noch vor dem letzten Fußbereich) bearbeitet.

Beispiel

Siehe data.

Siehe auch

data, footer, group, header, prologue, Bereichskommandos

4.5.4 Befehle in Datenbankjobs

4.5.4.1 Befehle in Datenbankjobs

Für Auswertung mit Datenbankjobs bietet die Programmiersprache einige spezielle Befehle an:
In der folgenden Übersicht gibt (*) den Vorgabewert an, wenn kein Steuerkommando benutzt wird.

AB	Dynamischer Zeilenumbruch für den folgenden Absatz
AK	Veraltet: Abkürzung von Feldnamen verbieten/erlauben
CP	Neue Seite beginnen, wenn weniger als N Zeilen frei
DE	Aufsteigende Reihenfolge im Datenbankjob
DX	Horizontalen Abstand zu den Linien einstellen
DY	Vertikalen Abstand zu den Linien einstellen
EC	Abbruch des Datenbankjobs bei Fehlern ein-/ausschalten
EVL	Endpunkt einer vertikalen Linie definieren
FF	Seitenvorschub (nicht mehr unterstützt)
FL	Fußbereich auf letzter Seite nicht ausgeben
GC	Gruppenbereich nicht berücksichtigen
GP	Veraltet: Gruppenbereich am Anfang einer neuen Gruppe
HE	Erster Kopfbereich wird vor Prolog ausgegeben
HF	Kopfbereich auf erster Seite nicht ausgeben
HL	Horizontale Linie
MB	Unterer Seitenrand
MM	Einheit für Größenangaben setzen
MR	Rechter Seitenrand
MT	Leerzeilen am Seitenanfang
NB	Abbruch des Datenbankjobs zulassen
NL	Nächsten Zeilenumbruch unterdrücken
PA	Neue Seite beginnen
PO	Linker Seitenrand
PL	Veraltet: Seitenlänge festlegen
PN	Beginn der Seiten-Nummerierung festlegen
PS	ZSum und ZCount nur für Seite verwenden
PW	Veraltet: Seitenbreite festlegen
RW	Mehrspaltige Ausgabe
ST	Datenbankjob beenden
VL	Startpunkt einer vertikalen Linie

4.5.4.2 AB Steuerkommando

Syntax

```
AB
```

Kategorie

Datenbankjobs

Erklärung

Dynamischer Zeilenumbruch für den folgenden Absatz.

4.5.4.3 AK Steuerkommando

Syntax

```
AK n
```

Kategorie

Datenbankjobs

Erklärung

0	Abkürzung von Labels verboten, Vorgabe
1	Abkürzung von Labels erlaubt

Dieser Steuerbefehl erlaubt es aus Gründen der Rückwärtskompatibilität, statt der vollständigen Bezeichner Abkürzungen zu verwenden. Die Standard-Einstellung ist seit *TurboDB Studio* die Einstellung aus. Es wird dringend empfohlen, keine Abkürzungen zu verwenden, weil das Programm dadurch schlechter lesbar und fehleranfälliger wird.

4.5.4.4 Bold Prozedur

Syntax

```
Bold(Zahl: Integer)
```

Kategorie

Datenbankjobs

Erklärung

Bold liefert in jedem Fall eine leere Zeichenkette als Ergebnis. Interessant an dieser Funktion ist nur der Seiteneffekt, dass auf Fettdruck umgeschaltet wird, wenn das Argument 1 ist, und wieder auf Normaldruck, wenn es sich um 0 handelt. Aus diesem Grund findet *Bold* nur in Ausgabeformaten Verwendung.

Beispiel

(Ausgabeformat in Datenbankjob):

```
$("Sehr geehrter Herr " Bold(1) Name Bold(0))
```

Siehe auch

Italic

4.5.4.5 Calc Prozedur

Syntax

```
Calc(Ausdruck)
Wert(Ausdruck)
```

Kategorie

Datenbankjobs

Erklärung

Hier handelt es sich um eine Pseudo-Funktion, die nur in Ausgabeformaten Verwendung findet. Da in Ausgabeformaten nur Funktionen (neben Feldzugriffen und Zeichenketten) und keine Berechnungen zugelassen sind, müssen solche in die syntaktische Form einer Funktion erhalten. Genau das liefert die Funktion *Calc*.

Beispiel

(Ausgabeformat im Datenbankjob):

```
$(Calc(5*3 + t))
```

Hinweis: Eine Alternative zur Funktion *Calc* bietet die Definition einer parameterlosen Funktion, was zu einer übersichtlicheren Form des Ausgabeformats führt (vor allem wenn mehrfach der selbe Ausdruck verwendet wird).

```
.DEF Ergebnis=5 * 3 + t
...
$(Ergebnis)
$(SUM(Ergebnis))
```

4.5.4.6 CP Steuerkommando

Syntax

```
CP nn
```

Kategorie

Datenbankjobs

Erklärung

nn Neue Seite beginnen, wenn weniger als nn Zeilen frei

Damit kann man beispielsweise verhindern, dass ein mehrzeiliger Datensatz durch einen Seitenumbruch getrennt wird. Im folgenden Beispiel wird vor der Ausgabe jedes Datensatzes geprüft, ob noch mindestens drei Zeilen auf der Seite verfügbar sind. Wenn nicht, wird ein Seitenvorschub durchgeführt und mit der Ausgabe des Datensatzes erst auf der folgenden Seite begonnen.

Beispiel

```
REPOPT
.DATEN
.CP 3
$Vorname $Name
$Straße
$PLZ $Ort
```

4.5.4.7 DE Steuerkommando

Syntax

```
DE n
```

Kategorie

Datenbankjobs

Erklärung

Normalerweise erfolgt die Dateiausgabe bezüglich des Zugriffs aufsteigend. Mit diesem Steuerkommando kann man die Reihenfolge umkehren.

0 (*) Aufsteigende Reihenfolge im Datenbankjob

1 Absteigende Reihenfolge im Datenbankjob

Beispiel

Liste nach Alter, die ältesten zuerst:

```
.PROLOG
.ZUGRIFF GEB-TAG.IND
.DE 1
.DATEN
$(Name, Vorname:40 Geboren:12)
```

4.5.4.8 DX Steuerkommando

Syntax

```
DX nnn
```

Kategorie

Datenbankjobs

Erklärung

Legt den horizontalen Abstand zwischen senkrechten Linien und Ausgabetext fest. Wenn eine vertikale Linie gezeichnet wird, bleiben links und rechts von ihr jeweils DX/2 Einheiten frei, bevor wieder Text ausgegeben wird.

Verwenden Sie den Befehl VX um bei der Ausgabe der Daten diesen Abstand zu überspringen.

Siehe auch

HL, VL, VX

4.5.4.9 DY Steuerkommando

Syntax

```
DY nnn
```

Kategorie

Datenbankjobs

Erklärung

Legt den vertikalen Abstand von Linien fest.

Siehe auch

HL, VL

4.5.4.10 do Kommando

Syntax

```
.do <Ausdruck>
```

Kategorie

Datenbankjobs

Erklärung

Das Kommando *do* erlaubt die Auswertung von beliebigen Ausdrücken auch in Datenbankjobs. Hier muss jede Zeile, die keinen Text enthält, durch ein Kommando gekennzeichnet sein. In Modulen wird das Kommando ignoriert. In der DOS-Version heißt dieses Kommando *Starte*.

Warnung

Mit Hilfe dieses Kommandos können auch Befehle ausgeführt werden, welche die korrekt Abarbeitung des Datenbankjobs verhindern. Zum Beispiel mit *do Access(...)* oder mit *do SetMark(...)*. Deshalb sollten Sie dieses Kommando mit größter Vorsicht verwenden und normalerweise keine Datenbank-Befehle damit ausführen.

Beispiel

```
do Input("Geben Sie das Startdatum ein", "Bereichsfestlegung")
```

4.5.4.11 EVL Steuerkommando

Syntax

```
EVL
```

Kategorie

Datenbankjobs

Erklärung

EVL steht für End Vertical Lines und definiert den Endpunkt einer vertikalen Linie. Alle noch nicht gezeichneten vertikalen Linien werden bis zur letzten horizontalen Linie gezeichnet. Wenn noch keine horizontale Linie gezeichnet wurde, dann bis zur aktuellen Ausgabeposition in Y-Richtung.

Alle *VL*-Befehle sind damit abgearbeitet und die Startpositionen werden gelöscht.

Beispiel

Siehe *VX* sowie das Beispiel-Projekt *Datenbankjobs*.

Siehe auch

VL

4.5.4.12 exit Kommando

Syntax

```
exit
```

Kategorie

Datenbankjobs

Erklärung

Das Kommando *exit* kann nur in Subreports eingesetzt werden. Es beendet dann die zugehörige Stufe des Unterreports. Die Bearbeitung wird hinter dem entsprechenden endsub fortgesetzt.

Beispiel

Es sollen maximal 10 passende Datensätze markiert werden:

```
sub KUNDEN.PLZ wie "8"
    if NMarks(KUNDEN) >= 10
       exit
    end
    SetMark(KUNDEN, RecNo(KUNDEN))
endsub
```

Siehe auch

Return, Halt, EndProg

4.5.4.13 FF Steuerkommando

Syntax

```
FF
```

Kategorie

Datenbankjobs

Erklärung

Erzeugt einen Befehl zum Seitenvorschub, wenn der Datenbankjob in eine Datei ausgegeben wird.

4.5.4.14 filter Kommando

Syntax

```
.filter Filterdefinition
```

Kategorie

Datenbankjobs

Erklärung

Explizite Filterdefinition

In den meisten Fällen liefert das AF (= automatischer Filter)-System des *TurboDB Studio* optimale Suchstrategien für alle Formen der Datenabfrage. Es gibt freilich Fälle, in denen der Anwender die Automatik überschreiben will, um eine noch effizientere Suche zu ermöglichen. Das trifft vor allem dann zu, wenn ein Filter über mehrere Hierarchiestufen eines Index gebildet werden kann. Das AF-System benutzt grundsätzlich nur die ersten beiden Stufen.

Eine explizite Filterdefinition besteht in der Angabe eines Indexbereiches. Bei der Bearbeitung der Tabelle wird der Startindex gesucht, und die Bearbeitung beginnt mit dem zugehörigen Datensatz. Falls auch ein Endindex angegeben wird, beendet *TurboDB Studio* die Arbeit, wenn ein Satz mit einer (ordnungsgemäß) späteren Indexinformation gelesen wird. Daraus ergibt sich die Voraussetzung für einen sinnvollen Filtereinsatz: Der Zugriff muss auf einem Index stehen, und die Bereichsangabe muss sich auf diesen Index beziehen.

Mit
```
.filter
```
wird der Filter aufgehoben.

Beispiel

Die Tabelle KUNDEN enthält die Felder "Land" und "PLZ". Es sollen alle deutschen Kunden im Postleitzahlbereich von 2 bis 4 bearbeitet werden. Es besteht ein Index LANDPLZ.IND mit der Beschreibung "Land, PLZ". Eine Selektion der Form "Land='D', PLZ von '2' bis '49999'" liefert zwar das gewünschte Ergebnis, benötigt allerdings unnötig viel Zeit, weil das AF-System nur die erste Hierarchiestufe (=Land) des Index verwendet, also sämtliche Kunden mit Land="D" zur Verfügung stellt. Hier die genauere Filterdefinition:

```
primtableis KUNDEN
setaccess LANDPLZ.IND
filter D,2/D,49999
```

Mit diesem Filter setzt die Bearbeitung direkt beim ersten Datensatz mit der angegebenen Bedingung auf und hebt nach dem letzten Satz auch wieder ab.

4.5.4.15 FL Steuerkommando

Syntax

```
FL n
```

Kategorie

Datenbankjobs

Erklärung

Bestimmt, ob der Fußbereich auch auf der letzten Seite gedruckt werden soll:

0	Kein Fußbereich auf der letzten Seite
1	Fußbereich auf der letzen Seite wird ausgegeben

4.5.4.16 font Kommando

Syntax

```
FONT SchriftartName = Schriftart, Größe, Auszeichnungen
```

Kategorie

Datenbankjobs

Erklärung

Definiert ein Format aus Schriftart, Schriftgröße und Auszeichnungen und gibt ihm einen Namen:

Schriftart	Name der Schriftart z.B. Arial oder Times New Roman
Größe	Schriftgröße in typographischen Punkten, meist 10 oder 12
Auszeichnungen	Kombination der Buchstaben k/i, f/b, u und n für kursiv/italic, fett/bold, unterstrichen/underlined und normal. Der Parameter ist nicht optional, darf also nicht weggelassen werden.

Das *font*-Kommando muss direkt hinter dem *report*-Kommando stehen, also nicht im Prolog, Kopf, oder einem anderen Abschnitt.

Beispiel

```
.REPORT
.FONT Üs2 = Arial,14,b
.FONT Std = Times New Roman,10,n
.FONT Hvh = Times New Roman,10,i
.DATEN
.SETFONT Üs2
Adresse von $Nachname, $Name

.SETFONT Std
$Straße
$(PLZ @Hvh$Ort@Std)
...
```

Siehe auch

setfont, InitFont

4.5.4.17 GC Steuerkommando

Syntax

```
GC n
```

Kategorie

Datenbankjobs

Erklärung

0 Bearbeitung des Gruppenbereiches nur im Datenbereich

1 Bearbeitung des Gruppenbereiches auch bei Subreports im Prolog, Vorgabe

Normalerweise wird der Gruppenbereich für die Ausgabe im Daten-Bereich ausgeführt. Mit diesem Steuerkommando kann man erreichen, dass der Gruppenbereich auch für Subreports im Prolog bearbeitet wird.

4.5.4.18 GP Steuerkommando

Veraltet, nicht mehr verwenden! Stattdessen können ab Visual Data Publisher 2 groupHeader und groupFooter eingesetzt werden.

Syntax

```
GP n
```

Kategorie

Datenbankjobs

Erklärung

0 Bearbeitung des Gruppenbereichs nach Gruppenwechsel, Vorgabe

1 Bearbeitung vor Gruppenwechsel

Siehe auch

G_Neu, G_alt, groupHeader, groupFooter

4.5.4.19 GetPara Prozedur

Syntax

```
GetPara(Zeichenkette: String): Integer
```

Kategorie

Datenbankjobs

Erklärung

Bei der übergebenen Zeichenkette muss es sich um einen gültigen Steuerbefehl (Punktkommando) handeln. Die Funktion liefert dann den aktuell gesetzten Wert für diesen Befehl. Ist nur im Datenbankjob sinnvoll.

Beispiel

```
GetPara('PL')    --> Seitenlänge
```

Falls nur noch maximal 5 Zeilen auf die Seite passen, soll ein Seitenumbruch durchgeführt werden.

```
.IF Line>=GetPara("PL")-5
.PA
.END
```

Siehe auch

SetPara, Steuerbefehle

4.5.4.20 GotoXY Prozedur

Syntax

```
GotoXY(X: Integer, Y: Integer)
```

Kategorie

Datenbankjobs

Erklärung

Setzt die aktuelle Druckposition auf die Koordinaten (x, y) in aktuellen Einheiten. Ist nur im Datenbankjob sinnvoll.

Beispiel

```
.MM 10
.Do GotoXY(10, 10)
```

Siehe auch

.MM, WhereX, WhereY

4.5.4.21 HE Steuerkommando

Syntax

```
HE n
```

Kategorie

Datenbankjobs

Erklärung

0 Erster Kopfbereich wird vor Prolog ausgegeben, Vorgabe

1 Kopfbereich erst vor erster Druckausgabe

Normalerweise wird der Prologbereich erst nach dem ersten Kopfbereich abgearbeitet. Wenn im Prolog aber Einstellungen stattfinden, die schon für den Kopfbereich gelten sollen, lässt sich mit diesem Steuerbefehl die Reihenfolge umkehren.

Beispiel

Bei folgendem Datenbankjob wird der Tabellenkopf vor der Gesamtüberschrift ausgegeben. Das lässt sich wie im zweiten Code-Stück zu sehen ändern, indem man die Reihenfolge der Abarbeitung umdreht:

```
.REPORT
.PROLOG
Liste der Mitglieder

.KOPF
Name:20 Vorname:20 Geburtsdatum:20
.DATEN
..hier kommen die Datensätze
```

So kann dies verhindert werden:

```
.REPORT
.HE 1
.PROLOG
Liste der Mitglieder

.KOPF
Name:20 Vorname:20 Geburtsdatum:20
.DATEN
..hier kommen die Datensätze
```

4.5.4.22 HF Steuerkommando

Syntax

```
HF n
```

Kategorie

Datenbankjobs

Erklärung

Bestimmt, ob der Kopfbereich auch auf der ersten Seite gedruckt werden soll:

0 Kein Kopfbereich auf der ersten Seite

1 Kopfbereich auf der ersten Seite wird ausgegeben (Vorgabe)

4.5.4.23 HL Steuerkommando

Syntax

```
.HL
.HL nnn
```

Kategorie

Datenbankjobs

Erklärung

Die erste Version zeichnet eine horizontale Linie von der aktuellen Druckposition bis zum rechten Rand. Wenn Startpunkte für vertikale Linien definiert sind, dann wird die horizontale Linie bis zur letzten vertikalen Linie gezeichnet.

Die zweite Version führt zu einer horizontalen Linie von der aktuellen Druckposition nach rechts mit der Länge von nnn Einheiten.

Siehe auch

VL

4.5.4.24 HT Steuerkommando

Syntax

```
HT n
```

Kategorie

Datenbankjobs

Erklärung

Wenn aktiviert, werden auch HTML-Formatierungen <b>, <i> und <Schriftart> im Datenbankjob als Eingabe akzeptiert.

4.5.4.25 include Kommando

Syntax

```
include Textdatei
```

Kategorie

Datenbankjobs

Erklärung

Einfügen eines Textbausteins in einen Datenbankjob.

Dem Kommando *include* wird der Name einer Textdatei nachgestellt. An dieser Stelle wird die Bearbeitung des momentanen Jobs unterbrochen und erst die im Argument des Kommandos spezifizierte Textdatei vom Datenträger gelesen und komplett bearbeitet. Nach deren Bearbeitung wird der ursprüngliche Text wieder aufgenommen. Einfügeanweisungen dürfen auch geschachtelt werden, das heißt, auch der einzufügende Text darf wiederum eine (oder mehrere) Einfügeanweisungen enthalten.

Das Einsatzgebiet ist das Einfügen von Textbausteinen in Datenbankjobs. Ein per *include* eingebundener Text kann auch *TurboPL*-Quelltext enthalten.

Die eingefügten Textbausteine dürfen keine eigene Textaufteilung (Prolog etc.) enthalten.

In Kopf- und Fußbereich ist dieses Kommando nicht angebracht, da sonst kein korrekter Seitenumbruch durchgeführt werden kann.

Das *include*-Kommando verarbeitet nur Quelltext. Im Gegensatz zu *ExecProg* wird das Kommando *include* während des Kompilierens und nicht während der Laufzeit ausgeführt.

Im einem Modul ist statt *include* das Kommando *USES* zu verwenden.

Beispiel

```
include Anreden.txt
include Vorbel.txt
```

Siehe auch

ExecProg, Uses

4.5.4.26 InitFont Prozedur

Syntax

```
InitFont(Definition: String)
```

Kategorie

Datenbankjobs

Erklärung

Definiert ein Format aus Schriftart, Schriftgröße und Auszeichnungen und gibt ihm einen Namen. Die zu übergebende Definition der Schriftart hat folgende Form: *Name = Schriftart, Größe, Auszeichnung*

Schriftart	Name der Schriftart z.B. Arial oder Times New Roman
Größe	Schriftgröße in typographischen Punkten, meist 10 oder 12
Auszeichnungen	Kombination der Buchstaben k/i, f/b und u für kursiv/italic, fett /bold und unterstrichen /unterlined.

Die Funktion entspricht dem Kommando font, kann aber die Schriftart dynamisch hinzufügen bzw. ändern.

Beispiel

```
InitFont("Aufzählung = Arial,12,kf")
```

Siehe auch

font

4.5.4.27 Italic Prozedur

Syntax

```
Italic(Value: Integer)
```

Kategorie

Datenbankjobs

Erklärung

Italic liefert in jedem Fall eine leere Zeichenkette als Ergebnis. Interessant an dieser Funktion ist nur der Seiteneffekt, dass auf Kursivdruck umgeschaltet wird, wenn das Argument 1 ist, und wieder auf Normaldruck, wenn es sich um 0 handelt. Aus diesem Grund findet *Italic* nur in Ausgabeformaten Verwendung.

Beispiel

```
$("Sehr geehrte Frau " Italic(1) Name Italic(0))
```

Siehe auch

Bold

4.5.4.28 let Kommando

Syntax

```
let <variablenname> = <wert>
var <variablenname> = <wert>
```

Kategorie

Datenbankjobs

Erklärung

Setzt die Variable auf den angegebenen Wert. Variablen, die mit *let* zugewiesen werden, müssen

nicht zuvor deklariert werden. *Let* hat als Kommando die selbe Funktion wie *var*. Eine mit let oder var definierte Variable hat keinen festen Typ, d.h. man kann ihr einmal eine Zahl, dann ein Datum und schließlich einen String zuweisen.

Beispiele

```
.var Gesamtertrag = Gesamtertrag + Teilertrag

.let Preis = Anzahl * Einzelpreis

.let Initialen = Links(Vorname, 1) + Links(Name, 1)

.falls Land='D'
.let Porto = 2.50
.ende
.falls Porto > 1
.var Briefmarken = Briefmarken + 1
.var Gesamtporto = Gesamtporto + Porto
.ende
```

Die Anrede in diesem Serienbrief soll nur einmal berechnet und dann verwendet werden:

```
.falls weiblich
.var Anrede = "Sehr geehrte Frau " + Name
.sonst
.var Anrede = "Sehr geehrter Herr " + Name
.ende
$Anrede,
es freut uns sehr, dass Sie, $Anrede unseren Prospekt angefordert haben.
```

4.5.4.29 MB Steuerkommando

Syntax

```
MB nnn
```

Kategorie

Datenbankjobs

Erklärung

Setzt den unteren Seitenrand (*margin bottom*) in logischen Einheiten. Die Voreinstellung ist 0.

Beispiel

Siehe PO.

Siehe auch

PO, MT, MR, Maßangaben

4.5.4.30 MM Steuerkommando

Syntax

```
MM 0|nnn
```

Kategorie

Datenbankjobs

Erklärung

Stellt die Größeneinheit für nachfolgende Angaben ein. Eine Einheit ist entweder die durchschnittliche Zeichengröße der Schriftart oder nnn/10 Millimeter. Typische Angaben für nnn sind:

nnn	Bedeutung
0	Voreinstellung: Angaben in Vielfachen der durchschnittlichen Buchstabengröße
1	Angabe in Zehntelmillimetern
10	Angabe in Millimetern

Sie können theoretisch die Maßeinheit jederzeit verstellen. Empfehlenswert ist es jedoch, sie im Initialisierungsteil des Datenbankjobs einzustellen und dann so zu lassen.

Beispiel

So zeichnen Sie einen horizontalen Strich von genau 10 cm Länge:

```
.MM 10
.HL 100
```

So wird der Befehl MM normalerweise verwendet:

```
.report
.mm 10
.po 30 mt 20 mr 30 mb 20
.prolog
.Überschrift
.daten
...
```

Siehe auch

Maßangabe

4.5.4.31 MR Steuerkommando

Syntax

```
MR nnn
```

Kategorie

Datenbankjobs

Erklärung

Setzt den rechten Seitenrand (*margin right*) in logischen Einheiten. Die Voreinstellung ist 0.

Beispiel

Siehe PO.

Siehe auch

PO, MT, MB, Maßangaben

4.5.4.32 MT Steuerkommando

Syntax

```
MT nn
```

Kategorie

Datenbankjobs

Erklärung

Legt den oberen Rand (*margin top*) auf nn logische Einheiten fest. Voreinstellung ist 0.

Beispiel

Siehe PO.

Siehe auch

PO, MR, MB, Maßangaben

4.5.4.33 NL Steuerkommando

Syntax

```
NL
```

Kategorie

Datenbankjobs

Erklärung

Kein Zeilenumbruch nach der folgenden Ausgabezeile

4.5.4.34 PA Steuerkommando

Syntax

```
PA
```

Kategorie

Datenbankjobs

Erklärung

Der Steuerbefehl bewirkt einen Seitenumbruch. Alle folgenden Ausgaben werden auf der nächsten Seite ausgedruckt.

4.5.4.35 PL Steuerkommando

Syntax

```
PL nn
```

Kategorie

Datenbankjobs

Erklärung

Netto-Seitenlänge auf nn Zeilen beschränken

Wird nicht mehr unterstützt, bitte verwenden Sie .MB.

4.5.4.36 PN Steuerkommando

Syntax

```
PN nn
```

Kategorie

Datenbankjobs

Erklärung

SeitenNummerierung bei nn beginnen

Mit diesem Steuerkommando lässt sich die automatische Seitennummerierung über die Systemvariable *$Seite* beeinflussen. *PN* setzt *$Seite* auf einen beliebigen Anfangswert. Damit kann man bei mehrseitigen Serienbriefen beispielsweise erreichen, dass bei jedem Datensatz die Seiten-Nummerierung von 1 beginnt:

Beispiel

```
.KOPF
                                                - $Seite -
.DATEN
..    mehrseitiger Serienbrief
.PN 0
.EPILOG
```

4.5.4.37 PO Steuerkommando

Syntax

```
PO nn
```

Kategorie

Datenbankjobs

Erklärung

Linken Rand auf nn logische Einheiten festlegen (page offset). Die Voreinstellung ist 0. Dieser Befehl wird gewöhnlich im Initialisierungsteil des Datenbankjobs verwendet.

Beispiel

Die Seitenränder auf links 5 Einheiten, oben 3 Einheiten, rechts 5 Einheiten und unten 3 Einheiten festlegen:

```
.PO 5, MT 3, MR 5, MB 3
```

Siehe auch

MT, MR, MB, Maßangaben

4.5.4.38 PS Steuerkommando

Syntax

```
PS
```

Kategorie

Datenbankjobs

Erklärung

Legt fest, dass sich *ZSUM* und *ZCOUNT* auf die aktuelle Seite beziehen.

Anmerkung

In *TurboDB Studio* gibt es die indizierten statistischen Funktionen um Summen, Mittelwerte usw. über einzelne Gruppen zu bilden. Summe und Anzahl in Gruppen war in früheren Versionen der Turbo Datenbank die Aufgabe von *ZSUM* und *ZCOUNT*. Mit dem Steuerbefehl PS wird festgelegt, dass sich *ZSUM* und *ZCOUNT* auch dann auf die aktuelle Seite beziehen, wenn Gruppen existieren.

4.5.4.39 PW Steuerkommando

Wird nicht mehr unterstützt, bitte verwenden Sie .MR.

4.5.4.40 RW Steuerkommando

Syntax

```
RW nn
```

Kategorie

Datenbankjobs

Erklärung

Ausgabe in nn Spalten, Voreinstellung ist 1.

4.5.4.41 ST Steuerkommando

Syntax

```
ST
```

Kategorie

Datenbankjobs

Erklärung

Datenbankjob beenden

Das Steuerkommando *ST* ist vorwiegend für den Einsatz in Datenbankjobs gedacht. In TurboPL -Modulen sollte man besser die Funktion *Halt* einsetzen.

4.5.4.42 primtableis Kommando

Syntax

```
primtableis <Table>
primärdatei <Tabelle>
```

Kategorie

Datenbankjobs

Erklärung

Mit dem Kommando *primtableis* oder *primärdatei* wird die Primärtabelle, d.h. die Tabelle für die Hauptdaten eines Datenbankjobs eingestellt. Mit *primTableIs* legt man sich anders als mit *PrimFile* schon beim Übersetzen auf die Primärtabelle fest. *primTableIs* kann nur im Prolog eines

Datenbankjobs aufgerufen werden.

Beispiel

```
.report
.prolog
.primärdatei KUNDEN
.daten
$Name, $Vorname
```

Siehe auch

PrimFile

4.5.4.43 relation Kommando

Syntax

```
relation <Relationsdefinition>
```

Kategorie

Datenbankjobs

Erklärung

Statische Links festlegen, Tabellen virtuelle öffnen und Inklusionen definieren

Das Relationskommando dient der Einrichtung von

- virtuellen Tabellen, über die ein zusätzlicher Zugriff auf bereits geöffnete Tabellen möglich wird, und
- statischen Links, mit denen die Tabellenverknüpfungen über das ADL-System erweitert oder überschrieben werden können, und
- Zwangsverknüpfungen, so dass immer eine Kombination mit der Primärdatei hergestellt wird, auch wenn kein verknüpfter Datensatz vorhanden ist (Erzwungene Bildung des Kreuzproduktes).

Die einzelnen Angaben werden einfach durch Komma getrennt hintereinander geschrieben. Die Länge eines Kommandos ist auf 255 Zeichen beschränkt. Jedes Relationskommando hebt das vorhergehende wieder auf.

Eine Relationsdefinition wird innerhalb von Datenbankjobs und TurboPL-Modulen mit dem Kommando RELATION angegeben. In Datenbankjobs ist dieses Kommando nur im Prolog zulässig. In allen anderen Textbereichen wird die Bearbeitung mit der Meldung "Illegale Operation" abgebrochen. Datenbank-weite Relationsdefinitionen können über den Menüpunkt *Tabelle/Verknüpfungen...* im Datenmodellfenster angelegt werden.

Virtuelle Öffnung einer bereits geöffneten Tabelle

Form: Tabellenname = Tabellenname

Der Tabellenname links vom Gleichheitszeichen darf nicht mit dem Tabellennamen einer geöffneten Tabelle übereinstimmen. Der Tabellenname rechts vom Gleichheitszeichen muss eine bereits (real oder virtuell) geöffnete Tabelle bezeichnen.

Beispiel

```
RELATION A1 = ADRESSEN
```

In der Folge kann auf die virtuell geöffnete Tabelle A1 genauso wie auf real geöffnete Tabelle zugegriffen werden. Es ist allerdings nicht möglich, eine virtuell geöffnete Tabelle zur Primärtabelle zu machen. Die virtuelle Tabelle wird wieder automatisch geschlossen beim

- Wechsel der Primärdatei,
- Ändern der Relation,
- Ende des Datenbankjobs.

Die virtuelle Tabelle übernimmt von ihrem realen Pendant sämtliche Eigenschaften mit Ausnahme der statischen Links.

Durch die Öffnung virtueller Tabellen sind folgende Zusatzmöglichkeiten gegeben:

- Relation einer Tabelle zu sich selbst,
- Entflechtung von mehreren gleichzeitigen Verknüpfungen zwischen zwei Tabellen.

Es ist erlaubt, eine Tabelle mehrfach an eine andere anzubinden.

Beispiel

Die Tabelle PERSONEN speichert Schauspieler und Regisseure.

Die Struktur der Tabelle FILME verfügt über ein Koppelfeld Regisseur, sowie über ein Relationsfeld Darsteller.

Ist die Primärtabelle FILME, kann mit der oben genannten Konstruktion sowohl auf den jeweiligen Regisseur als auch auf die Darsteller zugegriffen werden (die hier gemeinsam in der Tabelle PERSONEN gespeichert sind). Beispiel für ein Ausgabeformat im Datenbankjob:

```
$Titel:20 $Regisseur.Vorname+" "+$Regisseur.Name:30
SUB(/$FILME.Darsteller.Vorname:1+"."+$FILME.Darsteller.Name)S:3//
```

Ausgabe:

Der Clou	George Roy	C.Durning
		P. Newman
		R.Redford
		R.Shaw
		R.Walston
Der Blaumilchkanal	Ephraim Kishon	N.Azikiri
		S.Friedman
		B.Zur

Ist jedoch als Primärdatei PERSONEN.DAT eingestellt, ist der Zugriff nicht mehr so einfach möglich. In diesem Fall wird dem Relationsfeld der Vorrang gegenüber dem Koppelfeld gegeben.

Bei mehreren Relationsfeldern erfolgt die Ankopplung immer über das erste Feld in der Struktur.

Die Lösung erfolgt in diesem Fall über die Verwendung einer virtuellen Tabelle mit der Definition eines statischen Links:

```
RELATION = REGIE = PERSONEN, $REGIE[Laufende_Nummer] = $FILME[Regisseur]
```

Ausgabeformat:

```
Name,Vorname:30! Titel:20 $REGIE.Vorname" "$REGIE.Name/
```

Ausgabe:

Redford,Robert	Der Clou	George Roy
	Der große Gatsby	Francis Coppola...
Newman,Paul	Der Clou	George Roy
		Butch Cassady ...

Statische Links

Mit statischen Links kann die Verknüpfung von Tabellen (neu) definiert werden. Statische Links treten dabei an die Stelle von Equate-Joins und sorgen für wesentlich schnellere Zugriffe.

Sie können sämtliche geöffneten Tabellen einschließen. ADL-Verknüpfungen werden durch statische Links nur vorübergehend inaktiviert.

Definierte Links bleiben nur solange aktiv, bis

- ein Wechsel der Primärtabelle stattfindet oder
- der Link eine Neudefinition erfährt.

In unserem Beispiel soll eine Liste der Regisseure mit den Titeln ihrer Filme ausgegeben werden:

```
RELATION $PERSONEN[Laufende_Nummer] = $FILME[Regisseur]
```

Ausgabe:

Coppola, Francis	Der große Gatsby
Kishon, Ephraim	Der Blaumilchkanal
Roy, George	Der Clou ...

Erzwungene Bildung des Kreuzprodukts

Normalerweise werden Datensätze (der Primärtabelle) in Verbindung mit verknüpften Tabellen nur dann ausgegeben, wenn in diesen Dateien passende Einträge vorliegen.

Durch Aufzählung der angekoppelten Tabellen können alle Datensätze mit einem leeren Datensatz verknüpft werden, für die kein passender Eintrag vorliegt. Dadurch gelangen auch diese Datensätze zur Ausgabe.

```
RELATION $PERSONEN[Laufende_Nummer] = $FILME[Regisseur], FILME
```

Ausgabeformat:

```
Name, Vorname:30 Titel
```

Ausgabe:

Coppola, Francis	Der große Gatsby
Kishon, Ephraim	Der Blaumilchkanal
Newman, Paul	
Redford, Robert	
Roy, George	Der Clou
Shaw, Robert	

Hinweis

Der Einsatz des Relationskommandos sollte fortgeschrittenen Anwendern vorbehalten bleiben.

4.5.4.44 selection Kommando

Syntax

```
selection <search-condition>
selektion <search-condition>
```

Kategorie

Datenbankjobs

Erklärung

Innerhalb von Modulen hat dieses Kommando keine Wirkung. Es kann aber in Verbindung mit Datenbankjobs eingesetzt werden, um dem äußersten Report eine Selektion zuzuweisen. Dem Datenbereich werden dann nur diejenigen Satzkombinationen übergeben, die die Selektion erfüllen. das Kommando ist nur im Prolog zulässig.

Interessante Möglichkeiten bestehen darin, dass die Selektion auf bereits definierte Variablen und Funktionen zugreifen kann. Damit lassen sich auch dynamische Selektionen realisieren.

Beispiel

```
selection KUNDEN.Land="D" AND KUNDEN.PLZ[1] IN ["7","8","9"]
```

Siehe auch

Filter

4.5.4.45 setAccess Kommando

Syntax

```
setAccess <IndexName>
Zugriff <IndexName>
```

Kategorie

Datenbankjobs

Erklärung

Die Reihenfolge, in der auf die Datensätze der Primärtabelle zugegriffen wird, kann mit diesem einfachen Kommando eingestellt werden. Im Datenbankjob ist das Kommando nur im Prolog zulässig. Die erlaubten Argumente des Kommandos *setAccess* sind:

Nummer	physikalische Reihenfolge
Markierung	nur markierte Datensätze in der zeitlichen Reihenfolge der Markierungen

Indexname jeder zur Primärtabelle gehörende Index. Ein benutzerdefinierter Index kann hier mit oder ohne die Dateiendung angegeben werden.

Beispiele

Wenn die Primärtabelle KUNDEN die Indexe *Kunden.id*, Kunden.inr, LandPlz.ind und Name.ind enthält, sind die folgenden Aufrufe von *setAccess* in einem Datenbankjob KUNDEN gültig:

```
setAccess Markierung
setAccess Kunden.inr
setAccess LandPlz
setAccess Name.ind
setAccess Kunden.id
```

Siehe auch

Access, IndName, IndNo, Sortierung, SortBy

4.5.4.46 sortby Kommando

Syntax

```
sortby <index definition>
sortby <Indexdefinition>
```

Kategorie

Datenbankjobs

Erklärung

Dieses Kommando legt die Sortierung der Datensätze im Datenbereich eines Datenbankjobs fest. *Indexdefinition* ist eine Index-Beschreibung und muss nicht zu einem existierenden Index passen. Wenn kein Index dafür existiert, wird temporär einer angelegt.

Einsatz

Im Prolog eines Datenbankjobs

Beispiel

Dieser Datenbankjob gibt Adressdaten sortiert nach Name und Vorname aus, auch wenn dafür kein Index existiert.

```
.prologue
.sortby Name,Vorname
.data
$Vorname $Name
$Strasse
$PLZ $Ort
```

Siehe auch

SetAccess

4.5.4.47 setfont Kommando

Syntax

```
setfont <SchriftartName>
```

Kategorie

Datenbankjobs

Erklärung

Aktiviert ein zuvor mit *font* definiertes Format.

Beispiel

Siehe *font*

4.5.4.48 SetPara Prozedur

Syntax

```
SetPara(Steuerbefehle: String)
```

Kategorie

Datenbankjobs

Erklärung

Anstelle eines Steuerkommandos, das als Punktbefehl in den Datenbankjob ausgedrückt wird, können die Parameter auch über diese Funktion eingestellt werden. Der Vorteil dabei ist, dass die Werte dynamisch, also zu Laufzeit festgelegt werden können. Das Ergebnis der Funktion ist immer 0.

Beispiel

```
SetPara("PL 65, EC 1")
```

Siehe auch

GetPara, Steuerbefehle

4.5.4.49 var Kommando

Siehe let.

4.5.4.50 VL Steuerkommando

Syntax

```
VL nnn
```

Kategorie

Datenbankjobs

Erklärung

VL im Ausgabformat führt zu einer vertikalen Linie von der aktuellen Druckposition bis

· zur letzten horizontalen Linie, die vor der Ausgabe von *EVL* ausgegeben wurde, bzw. bis

· zur aktuellen Ausgabezeile, in der das Ausgabeelement *EVL* enthalten ist.

VL führt also zunächst einmal nur dazu, dass die aktuelle Druckposition gespeichert wird. Erst durch die Ausgabe von EVL wird die vertikale Linie gezogen.

Der Steuerbefehl kann, durch Komma getrennt, auch mehrfach hintereinander angegeben werden. Das optionale Argument bei VL gib an, um wie viele Einheiten weiter rechts die nächste Linie kommen soll.

Beispiel

Dieses Ausgabeformat erstellt die Kopfzeile für eine tabellarische Liste mit senkrechten Linien zwischen den Spalten:

```
$(Bold(1) VL 'Name' C_Form 15 VL 'First Name' C_Form 12 VL 'E-Mail' C_Form 30
VL Bold(0))
```

Ein Beispiel für die Verwendung mit Spaltenbreite finden Sie unter *VX*.

Siehe auch

EVL, VX, HL

4.5.4.51 VX Steuerkommando

Syntax

```
VX
```

Kategorie

Datenbankjobs

Erklärung

Lässt im Ausgabeformat genau soviel Zwischenraum, wie von einer senkrechten Linie inklusive

des zugehörigen Abstands benötigt wird. Der Zwischenraum entspricht also genau dem mit *DX* einstellbaren und abfragbaren Maß. Mit dem *VX*-Befehl können Sie die Datenausgabe präzise in die Gitterlinien einpassen.

Beispiel

Diese Zeilen in einem Serienbrief definieren Kästchen um jedes Datenfeld:

```
.DATEN
.VL 50, VL 50, VL
.HL
$(VX ErstesFeld:50 VX ZweitesFeld:50 VX)
.HL
.EVL
```

Siehe auch

VL, VX, DX

4.5.4.52 WhereX Prozedur

Syntax

```
WhereX: Integer
```

Kategorie

Datenbankjobs

Erklärung

Ermittelt die X-Koordinate der aktuellen Druckposition in aktuellen Einheiten. Ist nur im Datenbankjob sinnvoll.

Beispiel

```
.if WhereX > 10
.do GotoXY(10, WhereY)
.end
```

Siehe auch

GotoXY, WhereY

4.5.4.53 WhereY Prozedur

Syntax

```
WhereY: Integer
```

Kategorie

Datenbankjobs

Erklärung

Ermittelt die Y-Koordinate der Druckposition in aktuellen Einheiten. Ist nur im Datenbankjob sinnvoll.

Beispiel

```
.if WhereY > 10
.do GotoXY(WhereX, 10)
.end
```

Siehe auch

GotoXY, WhereX

4.5.4.54 wohin Kommando

Dieses Kommando wird in *TurboDB Studio* nicht mehr unterstützt, da Sie hier bei jedem Ausdruck im Dialogfeld festlegen können, wohin die Ausgabe gehen soll. Mit der Oberflächenfunktion *Drucken* ist es möglich Datenbankjobs und Berichte von einer Prozedur aus zu starten. Im zweiten Parameter der Funktion kann das Ausgabeziel festgelegt werden.

Um Daten in eine Text-Datei zu schreiben, haben Sie mehrere Möglichkeiten:

1. Verfassen Sie einen Datenbankjob oder eine Liste, die die Daten in der gewünschten Form ausgibt. Wählen Sie *Ausführen/Drucken* im Menü des Datenfensters und selektieren Sie im

Drucken-Dialog die Option *In Datei*. Zusätzlich markieren Sie je nach Wunsch Windows-Text, DOS-Text oder HTML.

2. Verwenden Sie in einem Datenbankjob oder in einem Modul die Prozeduren *Rewrite*, *Write*, *Writeln* und *Close*, um eine Textdatei zu erstellen.

4.6 Benutzerschnittstelle

4.6.1 Oberflächen-Funktionen

Oberflächenfunktionen sind die Methoden der Applikation, der Datenfenster und der Steuerelemente. Sie bewirken wie der Name schon sagt eine Änderung an der Benutzerschnittstelle des Programms und sind oft gleichbedeutend mit dem Ausführen eines Menübefehls.

Einige Oberflächenfunktionen, nämlich *Sortierung*, *ShowRec*, *GetStars*, *PutStars*, *MarkierungSetzen*, *StarNum* und einige andere, haben eine sehr ähnliche Funktion wie die entsprechenden Datenbankfunktionen. Im Unterschied zu diesen wirken Sie jedoch auf ein ganz bestimmtes Formular und ihr Effekt ist deshalb sofort sichtbar. Eine ausführliche Darstellung dieses Unterschieds finden Sie in "Datenbank-Programmierung und Oberflächenprogrammierung".

Alle Methoden des Formulars gibt es auch für die Applikation, wo sie dann nicht von einem spezifischen Datenfenster sondern von dem gerade aktiven ausgeführt werden. Ein Aufruf von *DatensatzEditieren* in einem Applikations-Modul sucht zuerst das aktive Datenfenster und ruft dann wiederum dessen Methode *DatensatzEditieren* auf. Weitere Informationen hierzu enthält das Kapitel "Applikations-Module und Formular-Module".

Abbruch	Bricht die Ausführung eines modalen Datenfensters ab.
AlleMarkierungenEntfernen	Alle sichtbaren Markierungen entfernen
AnDasEndeBlättern	Im aktuellen Datenfenster die letzte Seite eines mehrseitigen Formulars anzeigen.
AnDenAnfangBlättern	Im aktuellen Datenfenster die erste Seite eines mehrseitigen Formulars anzeigen.
AnfangDerTabelle	Im aktuellen Datenfenster den ersten Datensatz anzeigen
Attach	Aktuelles Datenfenster an die Veränderungen in seiner Tabelle anpassen
Auffrischen	Aktuelles Datenfenster auffrischen
BildAuswählen	Dialog zur Auswahl einer Bilddatei
DateiAuswählen	Dialog zur Auswahl eines Dateinamens ausführen
DatenfensterSuchen	Liefert einen Verweis auf ein Datenfenster oder ein Maskenelement
DatensatzAnzeigen	Einen Datensatz anzeigen
DatensatzAuswählen	Einen Datensatz im aktiven Datenfenster auswählen.
DatensatzBetrachten	Aktuellen Datensatz im aktiven Datenfenster betrachten.
DatensätzeÄndern	Editiermodus im aktuellen Datenfenster ein- und ausschalten
DatensätzeBearbeiten	Ein Datenfenster für das Formular mit dem vollständigen Namen <Formularname> öffnen
DatensätzeBetrachten	Datensätze im aktiven Datenfenster betrachten.
DatensatzEditieren	Aktuellen Datensatz im aktiven Datenfenster editieren.
DatensätzeEditieren	Alle Datensätze im aktiven Datenfenster editieren
DatensätzeImportieren	Aufruf des Import Assistenten
DatensätzeExportieren	Aufruf des Export Assistenten
DatensätzeMarkieren	Datensätze im aktiven Datenfenster markieren
DatensatzLöschen	Den aktuellen Datensatz im aktuellen Datenfenster löschen
Drucken	Das Projektelement mit dem vollständigen Namen <Elementname> drucken

DruckerEinrichten	Dialogfenster zum Einrichten des Druckers anzeigen
EndeDerTabelle	Im aktuellen Datenfenster den letzten Datensatz anzeigen
EndProg	Für jeden Datensatz des aktuellen Datenfensters, der eine Bedingung erfüllt wird eine Ersetzung durchgeführt
ExecMacro	Prozeduren ausführen
ExecProg	TurboPL-Programm ausführen
Execute	Programm ausführen
FormularÖffnen	Ein Datenfenster öffnen
GetStars	Markierte Datensätze merken
GibModus	Modus im aktuellen Datenfenster bestimmen
GibSicht	Ermitteln ob sich das aktuelle Datenfenster im Tabellen- oder Formularmodus befindet
Input	Dialogfenster zu Eingabe einer Zeichenkette anzeigen
IsStar	Markierung im aktiven Datenfenster überprüfen
Kopplung	Wenn das aktuelle Datenfeld ein Koppel- oder Relationsfeld ist, ein modales Datenfenster der angekoppelten Tabelle zur Auswahl von Datensätzen anzeigen
MarkierteDatensätzeLöschen	Die markierten Datensätze des aktuellen Datenfensters löschen
MarkierungEntfernen	Markierung des aktuellen Datensatzes entfernen
MarkierungSetzen	Markierung des aktuellen Datensatzes setzen
MasterPasswort	Master-Passwort ändern
MediumPause	Unterbricht das Abspielen einer Media-Datei
MediumSpielen	Startet das Abspielen einer Medien-Datei
MediumStop	Beendet das Abspielen einer Media-Datei
Message	Mitteilung in Dialogfenster ausgeben
MitBedingung	Suche mit Bedingung durchführen und die angegebene Aktion ausführen
NächsteMarkierung	Im aktuellen Datenfenster den nächsten markierten Datensatz anzeigen
NächsterDatensatz	Im aktuellen Datenfenster den nächsten Datensatz anzeigen
NeueDatensätze	Mehrere neue Datensätze im aktiven Datenfenster eingeben.
NeueDatensätzeEingeben	Neueingabe von Datensätzen ein- und ausschalten.
NeuerDatensatz	Einen neuen Datensatz im aktiven Datenfenster eingeben.
NeueVerknüpfteDatensätzeEingeben	Ein modales Datenfenster öffnen und dort verknüpfte Datensätze für den aktuellen Datensatz eingeben
NeuenVerknüpftenDatensatzEingeben	Einen verknüpften Datensatz in einem modalen Datenfenster eingeben
OrderAuswählen	Dialog zur Auswahl eine Ordners ausführen
Pause	Wartet einige hundertstel Sekunden
PlaySound	Spielt einen Klang ab
PutStars	gemerkte Datensätze markieren
Schließen	Das aktuelle Fenster schließen
SeiteAnzeigen	Zeigt eine Seite eines mehrseitigen Formulars an.
SetzeAusgabeDatei	Datei als Druckziel festlegen
SetzeDrucker	Stellt den Drucker und die Papiergröße ein
SetzeTabZiel	Bestimmt, welches Formular-Feld bei [Tab] oder [Enter] als nächstes fokussiert wird.
SetzeSicht	Zwischen Formular- und Tabellenmodus wechseln
Sortierung	Die Sortierung im aktuellen Datenfenster auf den angegebenen Index stellen

StarNum	Anzahl der mit Sternchen gekennzeichneten Datensätze im aktuellen Datenfenster ermitteln.
StarteDialog	Formular als Dialog öffnen
Suchen	Schnelle Indexsuche
VerknüpfteDatensätze	Öffnet ein Datenfenster mit den verknüpften Datensätzen
VorherigeMarkierung	Im aktuellen Datenfenster den vorherigen markierten Datensatz anzeigen
VorherigerDatensatz	Im aktuellen Datenfenster den vorherigen Datensatz anzeigen
Wählen	Wählvorgang über Modem oder ISDN-Karte durchführen
WartenStart	Anzeige des Sanduhr-Kursors.
WartenStop	Zurückschalten vom Sanduhr-Kursor auf Normal-Kursor
Weiterblättern	Im aktuellen Datenfenster die nächste Seite eines mehrseitigen Formulars anzeigen
Weitersuchen	Die letzte Suche im aktuellen Datenfenster wiederholen
Zurückblättern	Im aktuellen Datenfenster die vorherige Seite eines mehrseitigen Formulars anzeigen

4.6.2 Projekt-Verwaltung

4.6.2.1 Projekt-Verwaltung

Zugriff auf Eigenschaften des Projekts und der Projektelemente.

4.6.2.2 GetCompleteObjectName

Syntax

```
GetCompleteObjectName(Object: Object): String
```

Kategorie

Oberflächenfunktion

Erklärung

Gibt den Objektnamen als String zurück. Dieser String kann dann von anderen Funktionen weiterverwendet werden.

Beispiel

Es soll der Bericht ADRESSEN.Adressliste gedruckt werden

```
procedure Adresslistedrucken
    vardef o: Object
    o := Project.ADRESSEN.Adressliste
    Run(GetCompleteObjectName(o))
endproc
```

Der Aufruf *Run("ADRESSEN.Adressliste")* würde natürlich zum selben Ergebnis führen. Der Vorteil bei der Verwendung von Objekten besteht in der Objektprüfung, die bei reiner String-Verwendung nicht stattfinden kann.

Würde der Name der Adressliste geändert, so wird in obigen Beispiel der Compiler den Fehler sofort entdecken, das einfache *Run("ADRESSEN.Adressliste")* bleibt hingegen, da syntaktisch korrekt, unentdeckt.

Siehe auch

GetFileName

4.6.2.3 GetFileName

Syntax

```
GetFileName(Object: Object): String
```

Kategorie

Oberflächenfunktion

Erklärung

Gibt den Dateinamen eines Objektes als String zurück. Dieser String kann dann von anderen Funktionen weiterverwendet werden.

Beispiel

Es soll eine Indexsuche über den selbsterstellten Index "Komplett" durchgeführt werden

```
procedure AdresseFinden(Suchname)
    vardef o: Object
    vardef IndexName: String
    o := Project.ADRESSEN.INDEX_Komplett
    IndexName := GetFileName(o)
    Find(IndexName, Suchname)
endproc
```

Über die Verwendung des Index-Objektes kann der Compiler bei Änderungen des Indexnamens den Fehler sofort erkennen, was bei reiner Verwendung des Dateinamens nicht erkannt werden kann.

Siehe auch

GetCompleteObjectName

4.6.2.4 Master-Passwort

Syntax

```
MasterPassword
Master-Paßwort
```

Kategorie

Oberflächenfunktion

Erklärung

Das Dialogfenster für das Master-Passwort wird angezeigt und das Master-Passwort nach den Benutzereingaben geändert.

Beispiel

```
Procedure Master-Passwort_ändern;
       MasterPasswort;
EndProc
```

4.6.2.5 EndProg

Syntax

```
EndProg
```

Erklärung

Im Entwickler-Modus entspricht der Aufruf der Funktion *EndProg* dem Menüpunkt *Datei/Beenden*.

Im User-Modus oder in der Programmvorschau wird die Applikation ohne weiter Abfrage beendet. Damit ist es dem Entwickler möglich, das Beenden der Anwendung selbst zu gestalten.

Beispiel

```
procedure Exit_Programm
    if Message("Möchten Sie diese wunderbare Anwendung wirklich schließen?",
"Programm beenden", 2) = 1
       EndProg;
    end;
endproc;
```

Siehe auch

Exit, Return

4.6.2.6 Project Variable

Syntax

```
Project
```

Erklärung

Über diese Variable kann auf die Eigenschaften des Projektes zugegriffen werden. Elemente des Projektes sind unter anderem die Familien und deren Elemente wiederum die Projektelemente wie Tabellen, Berichte etc.

4.6.2.7 SetKey

Syntax

```
SetzeTaste(Zeichenkette: String)
SetKey(Zeichenkette: String)
```

Erklärung

Schickt die übergebene Zeichenkette als Folge von Tastendrücken an die Anwendung.

Beispiel

```
SetKey('ABC')
```

Siehe auch

TestKeys

4.6.2.8 TestKeys

Syntax

```
TestKeys(Auswahl: String): String
```

Erklärung

Prüft, ob sich ein Zeichen aus dem String *Auswahl* als Tastaturereignis in der Windows-Botschaftsschlange befindet und liefert dieses als Rückgabewert. Andernfalls wird ein Leerstring zurückgegeben.

Beispiel

```
if TestKeys('aA')
    Message('Die A-Taste wurde gedrückt!')
end
```

Siehe auch

SetKey

4.6.3 Festgelegte Prozedurnamen

4.6.3.1 Festgelegte Prozedurnamen

Findet *TurboDB Studio* eine dieser Prozeduren in Ihren *TurboPL*-Modulen, wird die Prozedur ausgeführt.

OnOpenProject	Startprozedur für Desktop-Anwendungen
OnCloseProject	Prozedur beim Beenden von Desktop-Anwendungen
OnWWWQuery	Startprozedur für Internet-Anwendungen

4.6.3.2 OnOpenProject

Syntax

```
OnOpenProject
```

Erklärung

Startprozedur für Anwendungen.

Findet eine mit *TurboDB Studio* erzeugte Anwendung eine Prozedur mit diesem Namen, wird Sie unmittelbar nach Öffnen des Projektes ausgeführt. Die Prozedur muss sich dazu im ersten Modul

einer Tabelle befinden. Es können auch mehrere *OnOpenProject* Prozeduren in einem Projekt enthalten sein, sie werden einfach nacheinander abgearbeitet. Formular-Module werden dabei nicht berücksichtigt.

Beispiel

```
PROCEDURE OnOpenProject
      Message("Herzlich Willkommen in dieser Anwendung", "Hallo")
ENDPROC
```

Siehe auch

OnCloseProject

4.6.3.3 OnCloseProject

Syntax

```
OnCloseProject
```

Erklärung

Prozedur beim Beenden von Anwendungen.

Diese Prozedur funktioniert analog zu *OnOpenProject* und wird unmittelbar vor dem Beenden des Programmes abgearbeitet. Die Prozedur muss sich dazu im ersten Modul einer Tabelle befinden. Es können auch mehrere *OnCloseProject* Prozeduren in einem Projekt enthalten sein, sie werden einfach nacheinander abgearbeitet. Formular-Module werden dabei nicht berücksichtigt.

Beispiel

```
PROCEDURE OnCloseProject
   IF Day(Today)=1
      Message("Bitte denken Sie an die Datensicherung am Monatsanfang",
"Sicherung")
   ELSE
      Message("Vielen Dank für die Nutzung dieses Programmes", "Danke")
   END
ENDPROC
```

Siehe auch

OnOpenProject

4.6.4 Klasse Datenfenster

Objekte der Klasse Datenfenster stehen für eigenständige geöffnete Formulare oder für eingebettete Tabellen innerhalb eines Formulars.

4.6.4.1 Navigation im Datenfenster

4.6.4.1.1 Navigation im Datenfenster

Die Funktionen dieser Gruppe sind Methoden des Datenfensters und können deshalb auch mit einem vorangestellten Datenfenster-Objekt aufgerufen werden. Wenn kein Datenfenster-Objekt vorangestellt ist, bezieht sich die Funktion in einem Formular-Modul auf das ausführende Datenfenster und in einem anderen Modul auf das gerade aktive Datenfenster.

4.6.4.1.2 AnfangDerTabelle

Syntax

```
AnfangDerTabelle
TopOfTable
```

Kategorie

Oberflächenfunktion

Erklärung

Im aktuellen Datenfenster wird der erste Datensatz bezüglich der gerade eingestellten Sortierung angezeigt.

Entspricht dem Menüpunkt *Suchen/Gehe zu/Anfang der Tabelle* im Menü des Datenfensters.

Beispiel

Die folgenden Zeilen ermöglichen es dem Anwender bestimmte Datensätze in der Tabelle KFZ zu markieren. Anschließend wird die Anzeige auf die markierten Datensätze beschränkt.

```
IF MarkRecs('Markieren Sie die gewünschten Einträge')
    Access( KFZ, 'Markierung');
    Attach;
    AnfangDerTabelle;
END;
```

Siehe auch

EndeDerTabelle, NächsterDatensatz, VorherigerDatensatz, DatensatzAnzeigen

4.6.4.1.3 CurrentRecNo

Syntax

```
CurrentRecNo: Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Liefert die Datensatznummer des gerade im Formulars oder in einer Formulartabelle angezeigten Datensatzes.

Beispiel

Diese Zeilen ermitteln die Satznummer des in der eingebetteten Tabelle *DataGrid1* des Formulars *MEDIUM.Plattensammlung* angezeigten Datensatzes der Tabelle *TITEL* und synchronisieren die zugehörige Tabelle. Anschließend gilt *RecNo(TITEL) = TitleTable.CurrentRecNo*.

```
vardef TitleTable: DataWnd;
TitleTable := FindDataWnd('MEDIUM.Plattensammlung', 'DataGrid1');
ReadRec(TITEL, TitleTable.CurrentRecNo);
```

Siehe auch

ShowRec/DatensatzAnzeigen

4.6.4.1.4 DatensatzAnzeigen

Syntax

```
DatensatzAnzeigen(RecNr: Integer): Integer;
ShowRec(RecNo: Integer): Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktiven Datenfenster den Datensatz mit der Satznummer *RecNo* an. Im Erfolgsfall liefert diese Funktion die Datensatznummer zurück, andernfalls (z.B. wenn die Satznummer nicht existiert) liefert sie einen negativen Wert.

Spezielle Werte für *RecNo*:

-6	Nächste Markierung
-5	Vorherige Markierung
-4	Ende der Tabelle
-3	Nächster Datensatz
-2	VorherigerDatensatz
-1	Anfang der Tabelle

Beispiel

Die folgenden Befehle arbeiten alle Datensätze mit Sternchen im aktuellen Datenfenster ab:

```
VarDef r: Real;
..Erst mal zum ersten Datensatz gehen
r := DatensatzAnzeigen(-1);
..Falls nicht markiert, zum nächsten markierten
```

```
If not IsStar
    r := DatensatzAnzeigen(-6);
End
While r >= 0
    ..Hier tun Sie, was mit den markierten Datensätzen zu tun ist
    ..Und gehen dann zum nächsten markierten Datensatz
    r := DatensatzAnzeigen(-6);
End;
```

Siehe auch

AnfangDerTabelle, EndeDerTabelle, NächsterDatensatz, NächsteMarkierung, VorherigerDatensatz, VorherigeMarkierung

4.6.4.1.5 EndeDerTabelle

Syntax

```
EndeDerTabelle
BottomOfTable
```

Kategorie

Oberflächenfunktion

Erklärung

Im aktuellen Datenfenster wird der letzte Datensatz bezüglich der gerade eingestellten Sortierung angezeigt.

Entspricht dem Menüpunkt *Suchen/Gehe zu/Ende der Tabelle* im Datenfenster.

Beispiel

In ein Formular können Makro-Schalter zur Navigation durch die Tabelle plaziert werden. Der Schalter mit der Aufschrift "Letzter Datensatz" erhält dann als Makro den Aufruf von *EndeDerTabelle*.

Siehe auch

AnfangDerTabelle, NächsterDatensatz, VorherigerDatensatz

4.6.4.1.6 MitBedingung

Syntax

```
MitBedingung(Bedingung: String; Aktion, Umschalten: Integer): Integer
BySelection(Selection: String; Action, Prompt: Integer): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Sucht im aktuellen Datenfenster nach Datensätzen, die die Bedingung erfüllen und führt die angegebene Aktion aus. Die Werte für den Parameter Aktion sind:

0 Ersten passenden Datensatz selektieren
1 Alle passenden Datensätze markieren
2 Bei allen passenden Datensätzen Markierung setzen
3 Bei allen passenden Datensätzen Markierung entfernen

Die Werte für den Parameter Umschalten sind:

0 Nach der Suche wird ein Dialog angezeigt, mit der Frage ob auf die markierten Datensätze umgeschaltet werden soll oder nicht.
1 Es wird keine Meldung angezeigt, falls Datensätze markiert wurden, wird automatisch auf Markierung umgeschaltet.
2 Keine Meldung, keine Umschaltung.

Der Rückgabewert entspricht der Anzahl der gefundenen Datensätze.

Entspricht dem Menüpunkt *Suchen/Mit Bedingung* im Menü des Datenfensters.

Beispiel

```
procedure Suche_nach_PLZ
    T-Eingabe := '';
```

```
    if Input('PLZ', 'Kunden suchen')
       OpenForm('KUNDEN.Formular');
       MitBedingung('PLZ = ' + T-Eingabe);
    end;
endproc;
```

Siehe auch

Suchen

4.6.4.1.7 NächsteMarkierung

Syntax

```
NächsteMarkierung
NextStar
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktuellen Datenfenster den nächsten markierten Datensatz an.

Entspricht dem Menüpunkt *Suchen/Gehe zu/Nächste Markierung* im Menü des Datenfensters.

Siehe auch

VorherigeMarkierung

4.6.4.1.8 NächsterDatensatz

Syntax

```
NächsterDatensatz
NextRecord
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktuellen Datenfenster den nächsten Datensatz bezüglich der gerade gesetzten Sortierung an.

Entspricht dem Menüpunkt *Suchen/Gehe zu/Nächster Datensatz* im Menü des Datenfensters.

Beispiel

In ein Formular können Makroschalter zur Navigation durch die Tabelle plaziert werden. Der Schalter mit der Aufschrift "Nächster" erhält dann als Makro den Aufruf von *NächsterDatensatz*.

Siehe auch

AnfangDerTabelle, EndeDerTabelle, VorherigerDatensatz

4.6.4.1.9 Suchen

Syntax

```
Suchen(Index, Suchbegriff: String)
Find(Index, SearchCondition: String)
```

Kategorie

Oberflächenfunktion

Erklärung

Sucht über die Sortierordnung des Index nach dem ersten Auftreten des Suchbegriffs.

Beispiel

```
Suchen("KFZ.ID", "Opel Kadett");
```

Siehe auch

MitBedingung

4.6.4.1.10 VorherigeMarkierung

Syntax

```
VorherigeMarkierung
PrevStar
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktuellen Datenfenster den vorherigen markierten Datensatz an.

Entspricht dem Menüpunkt *Suchen/Gehe zu/Vorherige Markierung* im Menü des Datenfensters.

Siehe auch

NächsteMarkierung

4.6.4.1.11 VorherigerDatensatz

Syntax

```
VorherigerDatensatz
PrevRecord
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktuellen Datenfenster den vorherigen Datensatz bezüglich der gerade eingestellten Sortierung.

Entspricht dem Menüpunkt *Suchen/Gehe zu/Vorheriger Datensatz* im Menü des Datenfensters.

Beispiel

In ein Formular können Makroschalter zur Navigation durch die Tabelle platziert werden. Der Schalter mit der Aufschrift "Vorheriger" erhält dann als Makro den Aufruf von *VorherigerDatensatz*.

Siehe auch

AnfangDerTabelle, EndeDerTabelle, NächsterDatensatz

4.6.4.1.12 Weitersuchen

Syntax

```
Weitersuchen
FindNext
```

Kategorie

Oberflächenfunktion

Erklärung

Wiederholt die letzte Suchaktion.

Entspricht dem Menüpunkt *Suchen/Weitersuchen* im Menü des Datenfensters.

Siehe auch

Suchen

4.6.4.2 Oberflächenmarkierungen

4.6.4.2.1 Oberflächenmarkierungen

Oberflächenmarkierungen sind als Punkte am linken Rand der Gitterdarstellung zu erkennen. Mit ihrer Hilfe kann der Anwender gefundene Einträge erkennen oder selbst Datensätze auswählen.

4.6.4.2.2 Alle Markierungen entfernen

Syntax

```
AlleMarkierungenEntfernen[(KontrollAbfrage: Real)]
RemoveAllStars[(Validation: Real)]
```

Kategorie

Oberflächenfunktion

Erklärung

Alle sichtbaren Markierungen (= Sternchen) im aktiven Datenfenster werden entfernt.

Mit dem optionalen Parameter *KontrollAbfrage* wird eingestellt, ob vor dem Ausführen der Aktion eine Bestätigung durch den Benutzer erfolgen soll.

KontrollAbfrage = 0	Benutzerdialog wird ausgeführt
KontrollAbfrage = 1	Benutzerdialog wird unterdrückt

Diese Funktion entspricht dem Menüpunkt *Suchen/Alle Markierungen* entfernen.

Beispiel

siehe DatensätzeMarkieren

Siehe auch

MarkierungSetzen, MarkierungEntfernen, DatensätzeMarkieren, DatensatzAnzeigen, NächsteMarkierung, VorherigeMarkierung, IsStar, StarNum

4.6.4.2.3 GetStars

Syntax

```
GetStars(Stars: Integer[]): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Speichert die Satznummer der mit einer Oberflächenmarkierung (Sternchen) versehenen Datensätze im übergebenen Feld. Auf diese Weise kann man sich die Markierungen merken und bearbeiten, beziehungsweise später wiederherstellen. Falls das Array zu klein ist, um alle Markierungen aufzunehmen, wird die Größe entsprechend angepasst, dabei wird ein mehrdimensionales Array zu einen eindimensionalen Array gemacht. Zurückgegeben wird die Anzahl der Elemente im Array.

Beispiel

siehe *PutStars*

Siehe auch

PutStars

4.6.4.2.4 IsStar

Syntax

```
IsStar: Integer
```

Kategorie

Oberflächenfunktion

Erklärung

IsStar liefert 1, wenn der aktuelle Datensatz im aktiven Datensatz sichtbar (mit einem Sternchen) markiert ist, sonst 0.

Beispiel

```
IF IsStar = 1
      ..Tu was
END
```

Siehe auch

StarNum, MarkierungSetzen, MarkierungEntfernen, AlleMarkierungenEntfernen

4.6.4.2.5 MarkierteDatensätzeLöschen

Syntax

```
MarkierteDatensätzeLöschen[(KontrollAbfrage, Transaktion: Integer)]
DeleteStars[(Validation, Transaction: Integer)]
```

Kategorie

Oberflächenfunktion

Erklärung

Löscht alle markierten Datensätze des aktuellen Datenfensters.

Mit den optionalen Parametern *KontrollAbfrage* und Transaktion kann eingestellt werden, ob vor dem Ausführen der Aktion eine Bestätigung durch den Benutzer erfolgen soll, bzw. ob die Aktion im Transaktionsmodus durchgeführt wird:

KontrollAbfrage	0	Bestätigung einholen (Vorgabe)
KontrollAbfrage	1	Benutzerdialog unterdrücken
Transaktion	0	nicht im Transaktionsmoduls durchführen (Vorgabe)
Transaktion	1	Transaktionsmodus aktivieren

Entspricht dem Menüpunkt *Bearbeiten/Markierte Datensätze löschen* im Menü des Datenfensters.

Beispiel

```
IF MarkRecs('Markieren sie die Einträge, die gelöscht werden sollen')
      MarkierteDatensätzeLöschen(1, 0);
END;
```

Siehe auch

DatensatzLöschen, MarkierungSetzen, MarkierungEntfernen, AlleMarkierungenEntfernen, IsStar, StarNum

4.6.4.2.6 Markierung entfernen

Syntax

```
MarkierungEntfernen
RemoveStar
```

Kategorie

Oberflächenfunktion

Erklärung

Die sichtbare Markierung (=Sternchen) des aktuellen Datensatzes im aktiven Datenfenster wird entfernt.

Siehe auch

MarkierungSetzen, AlleMarkierungenEntfernen, MarkiertenDatensätzeLöschen, IsStar, StarNum

4.6.4.2.7 Markierung setzen

Syntax

```
MarkierungSetzen
PutStar
```

Kategorie

Oberflächenfunktion

Erklärung

Die sichtbare Markierung (=Sternchen) des aktuellen Datensatzes im aktiven Datenfenster wird gesetzt.

Entspricht dem Menüpunkt *Suchen/Markierung* setzen.

Siehe auch

MarkierungEntfernen, AlleMarkierungenEntfernen, MarkiertenDatensätzeLöschen, IsStar,

StarNum

4.6.4.2.8 PutStars

Syntax

```
PutStars(Stars: Integer[]): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Markiert im aktuellen Datenfenster die Datensätze deren Nummern im übergebenen Feld enthalten sind.

Beispiel

```
vardef Marks: Integer[0]
ActivateForm('KUNDEN.Fomular_Kunden')
MarkRecs('Markieren Sie ein paar Datensätze')
GetStars(Marks)
RemoveAllStars
Message('Jetzt sind alle Markierungen verschwunden')
PutStars(Marks)
Message('Jetzt sind sie wieder da!')
```

Siehe auch

GetStars, MarkierungSetzen

4.6.4.2.9 StarNum

Syntax

```
StarNum: Integer
```

Kategorie

Oberflächenfunktion

Erklärung

StarNum liefert die Anzahl der sichtbaren Markierungen im aktuellen Datenfenster. Die sichtbaren Markierungen sind diejenigen, die mit einem Sternchen angezeigt werden. Sie werden auf Oberflächen-Markierungen genannt. Deshalb ist *StarNum* eine Oberflächen-Funktion. *NMarks* dagegen liefert die Anzahl der internen Markierungen der Tabelle zurück und ist daher eine Tabellen-Funktion.

Beispiel

Die Reihenfolge im Datenfenster KFZ ist auf Satznummer eingestellt, fünf Datensätze haben ein Sternchen:

```
..Diese Box zeigt den Text 'Sichtbare Markierungen: 5' an
Message("Sichtbare Markierungen: " + Str(StarNum));
..In dieser Box steht 'Interne Markierungen: 0'
Message("Interne Markierungen: " + Str(NMarks(KFZ)))
..Nach der nächsten Zeile sind nur noch fünf Datensätze im Datenfenster
SetSortOrder("Markierung");
..Nun gibt die Box den Text 'Sichtbare Markierungen: 0' an
Message("Sichtbare Markierungen: " + Str(StarNum));
..Dafür haben wir nun fünf interne Markierungen
Message("Interne Markierungen: " + Str(NMarks(KFZ)))
```

Siehe auch

AlleMarkierungenEntfernen, IsStar, MarkierungEntfernen, MarkierungSetzen, NMarks

4.6.4.3 Mit Fenstern arbeiten

4.6.4.3.1 Mit Fenstern arbeiten

Meldungen ausgeben, Daten abfragen, Formulare öffnen und schließen, Dateneingabe steuern...

4.6.4.3.2 Abbruch

Syntax:

```
Abbruch
Cancel
```

Kategorie

Oberflächenfunktion

Erklärung

Wenn das Datenfenster modal ist, wird die Ausführung abgebrochen und das Datenfenster geschlossen, als hätte der Anwender auf den Abbruch-Schalter gedrückt.

Besonders wichtig ist der Einsatz von Abbruch im Zusammenhang mit dem Ausführen eines Formulars als Dialog mittels der Funktion *DialogAusführen*. In diesem Fall wird ein Makroschalter mit der Aufschrift "Abbruch" in das Formular eingebaut, wobei Abbruch als auszuführendes Makro anzugeben ist.

Siehe auch

StarteDialog, Schließen

4.6.4.3.3 AnDasEndeBlättern

Syntax

```
AnDasEndeBlättern
ViewLastPage
```

Kategorie

Oberflächenfunktion

Erklärung

Im aktuellen Datenfenster wird die letzte Seite eines mehrseitigen Formulars angezeigt.

Entspricht dem Menüpunkt *Ansicht/Seite/An das Ende* blättern im Menü des Datenfensters.

Beispiel

In einem mehrseitigen Formular können Makroschalter angebracht werden um das Umschalten zwischen den einzelnen Seiten zu ermöglichen. Der Schalter mit der Aufschrift "Zur letzten Seite" erhält als Makro den Aufruf von *AnDasEndeBlättern*.

Siehe auch

AnDenAnfangBlättern, Weiterblättern, Zurückblättern, SeiteAnzeigen

4.6.4.3.4 AnDenAnfangBlättern

Syntax

```
AnDenAnfangBlättern
ViewFirstPage
```

Kategorie

Oberflächenfunktion

Erklärung

Im aktuellen Datenfenster wird die erste Seite eines mehrseitigen Formulars angezeigt.

Entspricht dem Menüpunkt *Ansicht/Seite/An den Anfang* blättern.

Beispiel

In einem mehrseitigen Formular können Makroschalter angebracht werden um das Umschalten zwischen den einzelnen Seiten zu ermöglichen. Der Schalter mit der Aufschrift "Zur ersten Seite" erhält als Makro den Aufruf von *AnDenAnfangBlättern*.

Siehe auch

AnDasEndeBlättern, Weiterblättern, Zurückblättern, SeiteAnzeigen

4.6.4.3.5 AnzahlZeilen

Syntax

```
RowNum
AnzahlZeilen
```

Kategorie

Oberflächenfunktion

Erklärung

Liefert die Anzahl der angezeigten Datensätze im Datenfenster.

Beispiel

Demonstriert, wie nach einer Suche mit Umschaltung die Anzahl der gefundenen Datensätze ermittelt wird:

```
vardef W: DataWnd;
W := FindDataWnd('KUNDEN.EingabeFormular');
W.BySelection('Umsatz < 30000', 1, 1);
Message(W.RowNum + ' Datensätze wurden gefunden');
```

Siehe auch

Auffrischen

4.6.4.3.6 Attach

Syntax

```
Attach
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt die Einstellungen des aktiven Datenfensters auf die Werte der zugehörigen Tabelle. Diese Einstellungen sind:

- der aktuelle Datensatz
- die (interne) Markierungsliste und
- der aktuelle Zugriff.

Der Sinn von *Attach* besteht darin, dass das Datenfenster nach Änderungen in den Tabellen durch Datenbank-Befehle wie *ReadRec*, *SetMark*, *Access* usw. diese Änderungen auch übernimmt. Ohne einen Aufruf von *Attach* spiegeln sich die Auswirkungen von Datenbank-Befehlen nicht in den Formularen wieder.

Attach wird meist entweder ganz am Ende einer Prozedur aufgerufen, damit das Formular nach dem Ausführen der Prozedur alle Änderungen auf Datenbank-Ebene wiederspiegelt oder vor dem Aufruf einer modalen Datenfensterprozedur wie *EditRec* oder *ChooseRecs*.

Der Unterschied zu *Auffrischen* besteht darin, dass Attach Veränderungen durch Tabellenfunktionen "mitbekommt", *Refresh* dagegen nicht.

Beispiel

Ein Datenfenster für die Tabelle KFZ wird geöffnet, um diejenigen Datensätze anzuzeigen, die mit dem aktuellen Datensatz der Tabelle KUNDEN verknüpft sind. Dazu werden diese Datensätze in einem Aufruf von *LoopRecs* markiert und dann der Zugriff auf diese markierten Datensätze gestellt:

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
PrimFile(KUNDEN);
LoopRecs(KFZ, SetMark(RecNo(KFZ)));
PrimFile(KFZ);
Access(KFZ, "Markierung");
Attach;
```

Siehe auch

Refresh/Auffrischen

4.6.4.3.7 Auffrischen

Syntax

```
Auffrischen
Refresh
```

Kategorie

Oberflächenfunktion

Erklärung

Frischt die Darstellung im Datenfenster auf. Besonders nützlich, wenn durch Makros die Daten geändert wurden.

Im Gegensatz zu *Attach* werden durch *Refresh* der aktuelle Datensatz, die Markierungen und die Sortierung nicht verändert.

Beispiel

In der Eingabekontrolle eines Formulars soll unter "Ausführen beim Verlassen" der Inhalt des Feldes Kontostand geändert werden, wobei die Änderung sofort sichtbar sein soll.

```
Kontostand := Kontostand + Zugang;
Auffrischen;
```

Siehe auch

Attach

4.6.4.3.8 BildAuswählen

Syntax

```
BildAuswählen(Überschrift: String, Dateityp: Integer): Integer
ChoosePicture(Title: String, FileType: Integer): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt ein Dialogfenster für die Auswahl einer Bilddatei mit der Überschrift *Überschrift* an. Das selektierte Bild wird in einer Vorschau angezeigt.

In Dateityp kann einer der folgenden Dateitypen angegeben werden:

2	BMP
4	PCX
8	WAV
16	WMF
32	GIF
64	JPEG
127	Alle

Um eine Auswahl zu ermöglichen, können die einzelnen Zahlen auch addiert werden.

Die Vorbelegung des Dateinamens wird der Systemvariablen *T-Eingabe* entnommen. Dorthin wird auch der eingegebene Pfad zurückgeschrieben.

Der Rückgabewert beträgt 1, falls die Eingabe bestätigt wurde.

Beispiel

```
PROCEDURE AuswahlEinesJpgOderGifBildes: String
      T-Eingabe := "C:\VDP\DATEN\*.JPG";
      IF ChoosePicture("Bilddatei auswählen", 32 + 64) = 1
           RETURN T-Eingabe;
      ELSE
           RETURN "";
      END
ENDPROC
```

Siehe auch

Input ,OrdnerAuswählen , DateiAuswählen

4.6.4.3.9 DateiAuswählen

Syntax

```
DateiAuswählen(Überschrift: String [, Dateifilter: String [, var DateiName:
String]]): Integer
ChooseFile(Title: String [, FileFilter: String [, var FileName: String]]):
Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt ein Dialogfenster für die Dateiauswahl mit der Überschrift *Überschrift* an. In Dateifilter kann ein Filter für Dateitypen in der Form *'VDP-Tabelle|*.TDB|VDP-Formular|*.FRM'* angegeben werden (optional).

Falls die Variable *DateiName* angegeben ist, wird sie als Vorbelegung für den Dateinamen genommen und am Ende der Funktion mit dem ausgewählten Dateinamen belegt. Falls sie nicht angegeben ist, wird dafür die globale Variable *T-Eingabe* verwendet. Der Rückgabewert beträgt 1, falls die Eingabe bestätigt wurde.

Beispiel

```
procedure AuswahlEinerTDBDatei: String
    vardef DateiName: string;
    DateiName := "C:\VDP\DATEN\*.TDB";
    if ChooseFile("Tabelle auswählen", "VDP-Tabelle|*.dat|REL-Tabelle|*.rel|",
DateiName) = 1
        return DateiName;
    else
        return "";
    end;
endproc;
```

Siehe auch

Input, OrdnerAuswählen, BildAuswählen

4.6.4.3.10 DatensatzAnzeigen

Syntax

```
DatensatzAnzeigen(RecNr: Integer): Integer;
ShowRec(RecNo: Integer): Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktiven Datenfenster den Datensatz mit der Satznummer *RecNo* an. Im Erfolgsfall liefert diese Funktion die Datensatznummer zurück, andernfalls (z.B. wenn die Satznummer nicht existiert) liefert sie einen negativen Wert.

Spezielle Werte für *RecNo*:

-6	Nächste Markierung
-5	Vorherige Markierung
-4	Ende der Tabelle
-3	Nächster Datensatz
-2	VorherigerDatensatz
-1	Anfang der Tabelle

Beispiel

Die folgenden Befehle arbeiten alle Datensätze mit Sternchen im aktuellen Datenfenster ab:

```
VarDef r: Real;
..Erst mal zum ersten Datensatz gehen
r := DatensatzAnzeigen(-1);
```

```
..Falls nicht markiert, zum nächsten markierten
If not IsStar
    r := DatensatzAnzeigen(-6);
End
While r >= 0
    ..Hier tun Sie, was mit den markierten Datensätzen zu tun ist
    ..Und gehen dann zum nächsten markierten Datensatz
    r := DatensatzAnzeigen(-6);
End;
```

Siehe auch

AnfangDerTabelle, EndeDerTabelle, NächsterDatensatz, NächsteMarkierung, VorherigerDatensatz, VorherigeMarkierung

4.6.4.3.11 DatenfensterSuchen

Syntax

```
DatenfensterSuchen(Formular[, Maskenelement]: String): Object
FindDataWnd(Form[, Maskelement]: String): Object
```

Kategorie

Oberflächenfunktion

Erklärung

DatenfensterSuchen sucht ein geöffnetes Datenfenster für das Formular Formular. Der zweite Parameter *Maskenelement* ist optional. Wenn Sie ihn nicht angeben, liefert *DatenfensterSuchen* den Verweis auf das gefundene Datenfenster zurück, bzw. den Wert *Null*, wenn kein passendes Datenfenster vorhanden ist.

Um auf eine eingebettete Tabelle zuzugreifen verwenden Sie den zweiten Parameter *Maskenelement*. Damit können Sie innerhalb des Datenfensters ein einzelnes Maskenelement über seinen Namen ansprechen. Dieser Name wird im Formulareditor in den Eigenschaften des Elements eingetragen.

- Der Formularname muss vollständig angegeben sein, also in der Form <*TABELLE.Formularname*>
- Momentan können nur eingebettete Tabellensichten als Maskenelement angesprochen werden.

Beispiel

```
VarDef Datenfenster: DataWnd;
Datenfenster := DatenfensterSuchen("KUNDEN.Formular_KUNDEN");
IF Assigned(Datenfenster)
      Datenfenster.AlleMarkierungenEntfernen(1);
END
VarDef EingebetteteTabelle: Object;
EingebetteteTabelle := DatenfensterSuchen("KUNDEN.Fomular_KUNDEN",
"Fahrzeuge");
IF Assigned(EingebetteteTabelle)
      EingebetteteTabelle.NächsterDatensatz
END
```

Siehe auch

Programmieren mit Objekten

4.6.4.3.12 DatensatzAuswählen

Syntax

```
DatensatzAuswählen(Kommentar: String): Integer
ChooseRec(Comment: String): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet das aktive Datenfenster auf modal, zeigt *Comment* im Panel an und sperrt alle Befehle außer den zur Auswahl eines Datensatz nötigen. Der Benutzer kann einen Datensatz selektieren und dann bestätigen oder abbrechen. Der Rückgabewert ist die ausgewählte Satznummer bzw. 0

bei Abbruch.

Beispiel

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
vardef SatzNummer: Integer;
SatzNummer := DatensatzAuswählen("Wählen Sie das gewünschte Fahrzeug aus.");
if SatzNummer > 0
      ReadRec(KFZ, SatzNummer);
      Message("Dieses Fahrzeug wird von der Firma " + Hersteller + "
geliefert.");
end
Schließen
```

Siehe auch

NeuerDatensatz, NeueDatensätze, DatensatzEditieren, DatensätzeEditieren, ExecModal, DatensätzeMarkieren, DatensatzBetrachten, DatensätzeBetrachten, NeueVerknüpfteDatensätzeEingeben, VerküpfteDatensätze

4.6.4.3.13 DatensatzBetrachten

Syntax

```
DatensatzBetrachten(Kommentar: String)
ViewRec(Comment: String)
```

Kategorie

Oberflächenfunktion

Erklärung

DatensatzBetrachten schaltet im aktuellen Datenfenster alle die Befehle aus, die nicht zum Betrachten des aktuellen Datensatzes benötigt werden und wechselt in den modalen Modus. Der Anwender kann den aktuellen Datensatz betrachten und dann den OK-Schalter betätigen.

Beispiel

```
DatensätzeBearbeiten("KUNDEN.Kundenkartei");
DatensatzAnzeigen(312);
DatensatzBetrachten("Dies ist der Kunde mit der Satznummer 312");
Schließen
```

Siehe auch

NeuerDatensatz, NeueDatensätze, DatensatzAuswählen, DatensatzEditieren, DatensätzeEditieren , DatensätzeMarkieren, DatensätzeBetrachten, ExecModal, NeueVerknüpfteDatensätzeEingeben, VerknüpfteDatensätze

4.6.4.3.14 DatensätzeÄndern

Syntax

```
DatensätzeÄndern
ModifyRecords
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet im aktuellen Datenfenster die Editiermöglichkeit ein und aus. Entspricht dem Menüpunkt *Bearbeiten/Datensätze* ändern.

Beispiel

In einem Formular wird ein Makroschalter mit der Aufschrift Ändern platziert. Als Makro erhält er den Aufruf von *DatensätzeÄndern*

Beim ersten Betätigen des Schalters wird der Editiermodus ein- und beim nächsten mal ausgeschaltet.

Siehe auch

NeueDatensätzeEingeben, DatensatzLöschen, NeueVerknüpfteDatensätzeEingeben, DatensätzeBearbeiten

4.6.4.3.15 DatensätzeBearbeiten

Syntax

```
DatensätzeBearbeiten(Formular: String)
ActivateForm(Form: String)
```

Kategorie

Oberflächenfunktion

Erklärung

Öffnet ein Datenfenster für das angegebenen Formular. Form muss der vollständige Name des Formulars sein, also Tabellenname, Punkt, Formularname. Um ein zusätzliches Formular zu öffnen, benutzen Sie die Oberflächen-Funktion FormularÖffnen.

Beispiel

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
```

Siehe auch

DatensätzeÄndern, FormularÖffnen, Schließen

4.6.4.3.16 DatensätzeBetrachten

Syntax

```
DatensätzeBetrachten(Kommentar: String)
ViewRecs(Comment: String)
```

Kategorie

Oberflächenfunktion

Erklärung

DatensätzeBetrachten schaltet im aktuellen Datenfenster alle die Befehle aus, die nicht zum Betrachten der Datensätze benötigt werden und wechselt in den modalen Modus. Der Anwender kann die Datensätze der Tabelle betrachten und mit OK den modalen Modus verlassen.

Beispiel

```
DatensätzeBearbeiten("KUNDEN.Kundenkartei");
DatensätzeBetrachten("Blättern Sie durch die Kundenkartei.");
Schließen
```

Siehe auch

NeuerDatensatz, NeueDatensätze, DatensatzAuswählen, DatensatzEditieren, DatensätzeEditieren , DatensätzeMarkieren, DatensatzBetrachten, ExecModal, NeueVerknüpfteDatensätzeEingeben, VerknüpfteDatensätze

4.6.4.3.17 DatensatzEditieren

Syntax

```
DatensatzEditieren
EditRec(Kommentar: String): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet das aktive Datenfenster auf modal, zeigt *Kommentar* im Panel an und sperrt alle Befehle außer den zum Ändern des aktuellen Datensatzes nötigen. Der Benutzer kann den aktuellen Datensatz ändern und dann bestätigen oder abbrechen. Der Rückgabewert ist die ausgewählte Satznummer bzw. 0 bei Abbruch.

Beispiel

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
AnDenAnfangBlättern;
EditRec("Ändern Sie die Angaben.");
Schließen
```

Siehe auch

NeuerDatensatz, NeueDatensätze, DatensatzAuswählen, DatensätzeEditieren,

DatensatzBetrachten, DatensätzeBetrachten, ExecModal, DatensätzeMarkieren, NeueVerknüpfteDatensätzeEingeben, VerknüpfteDatensätze

4.6.4.3.18 DatensätzeEditieren

Syntax

```
DatensätzeEditieren(Kommentar: String): Integer
EditRecs(Comment: String): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet das aktive Datenfenster auf modal, zeigt *Kommentar* im Panel an und sperrt alle Befehle, die nicht zum Editieren der Datensätze benötigt werden. Der Benutzer kann die Datensätze ändern und die modale Bearbeitung mit OK beenden. Der Rückgabewert ist die zuletzt angezeigte Satznummer bzw. 0 bei Abbruch.

Beispiel

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
AnDenAnfangBlättern;
DatensätzeEditieren("Geben Sie die aktuellen Daten der Fahrzeuge ein.");
Schließen
```

Siehe auch

NeuerDatensatz, NeueDatensätze, DatensatzAuswählen, DatensatzEditieren, DatensatzBetrachten, DatzeBetrachten, ExecModal, DatensätzeMarkieren, NeueVerknüpfteDatensätzeEingeben, VerknüpfteDatensätze

4.6.4.3.19 DatensätzeImportieren

Syntax

```
DatensätzeImportieren: Integer
ImportRecords: Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Aufruf des Import Assistenten. Die Datensätze werden in die Tabelle des aktiven Datenfensters importiert.

Beispiel

```
Procedure ImportRecordsIntoKFZTable;
    ActivateForm("KFZ.Fahrzeuge");
    ImportRecords;
EndProc;
```

Siehe auch

ImportODBC, ExportRecords

4.6.4.3.20 DatensätzeExportieren

Syntax

```
DatensätzeExportieren
ExportRecords
```

Kategorie

Oberflächenfunktion

Erklärung

Aufruf des Export Assistenten. Die Datensätze werden aus der Tabelle des aktiven Datenfensters exportiert.

Beispiel

```
Procedure ExportKFZRecords;
```

```
    ActivateForm("KFZ.Fahrzeuge");
    ExportRecords;
EndProc;
```

Siehe auch

ImportRecords

4.6.4.3.21 DatensatzLöschen

Syntax

```
DatensatzLöschen([KontrollAbfrage: Integer])
DeleteRecord([Validation: Integer])
```

Kategorie

Oberflächenfunktion

Erklärung

Löscht den aktuellen Datensatz im aktuellen Datenfenster. Mit dem optionalen Parameter *KontrollAbfrage* kann die Abfrage an den Benutzer unterdrückt werden, ob der Datensatz wirklich gelöscht werden soll.

KontrollAbfrage = 0	Benutzerdialog wird ausgeführt (Vorgabe)
KontrollAbfrage = 1	Benutzerdialog wird unterdrückt

Entspricht dem Menüpunkt Bearbeiten/Datensatz löschen im Menü des Datenfensters.

Beispiel

Die folgenden Zeilen löschen den neunten Datensatz der Tabelle des aktuellen Datenfensters:

```
ShowRec(9);
DatensatzLöschen(1);
```

Siehe auch

DatensätzeÄndern, NeueDatensätzeEingeben, NeueVerknüpfteDatensätzeEingeben, MarkierteDatensätzeLöschen

4.6.4.3.22 DatensätzeMarkieren

Syntax

```
DatensätzeMarkieren(Kommentar: String): Integer
MarkRecs(Comment: String): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet das aktive Datenfenster auf modal, zeigt *Kommentar* im Panel an und sperrt alle Befehle außer den zur Markierung von Datensätzen nötigen. Der Benutzer kann beliebig viele Datensätze markieren und dann bestätigen oder abbrechen. Der Rückgabewert ist größer 0, falls die Markierung bestätigt wurde, bzw. 0 bei Abbruch.

Beispiel

```
..Eventuell vorhandene Markierungen entfernen
AlleMarkierungenEntfernen(1);
..Datenfenster modal schalten
if DatensätzeMarkieren("Markieren Sie die gewünschten Einträge.") > 0
   Message("Sie haben " + Val(StarNum) + " Fahrzeuge ausgewählt.");
end
```

Siehe auch

NeuerDatensatz, NeueDatensätze, DatensatzAuswählen, DatensatzEditieren, DatensätzeEditieren, ExecModal, DatensatzBetrachten, DatensätzeBetrachten, NeueVerknüpfteDatensätzeEingeben, VerknüpfteDatensätze, AlleMarkierungenEntfernen

4.6.4.3.23 DruckerEinrichten

Syntax

```
DruckerEinrichten
SetupPrinter
```

Kategorie

Oberflächenfunktion

Erklärung

Das Dialogfenster zum Einrichten des Druckers wird angezeigt.

Entspricht dem Menüpunkt *Datei/Drucker* einrichten.

Beispiel

Vor dem Ausdruck des Datenbankjobs anschreiben der Tabelle KUNDEN wird der Dialog zum Einrichten des Druckers angezeigt.

```
DruckerEinrichten;
Drucken('KUNDEN.Anschreiben');
```

Siehe auch

Drucken SetzeAusgabeDatei

4.6.4.3.24 ExecMacro

Syntax

```
ExecMacro(Modulname: Datei, Befehl1: Ausdruck, Befehl2: Ausdruck, ...): ?
```

Kategorie

Oberflächenfunktion

Erklärung

ExecMacro wird nicht mehr benötigt und ist deshalb nicht mehr empfohlen. Seit *TurboDB Studio* sind immer alle Module des Projekts im Speicher und müssen nicht nachgeladen werden. Falls Sie in bestehenden Modulen diese Funktion verwenden, können Sie sie in den meisten Fällen weiter benutzen. Besser ist es aber, es durch einen direkten Funktionsaufruf zu ersetzen.

Bestehende Aufrufe von ExecMacro entfernen

Aufruf von *ExecMacro* in einem Modul: *ExecMacro* war noch nie für den Einsatz in Modulen gedacht. Rufen Sie die Prozedur(en) direkt auf und fügen sie das entsprechende Module mit *uses* dem aufrufenden Modul hinzu. Sie gewinnen dabei auch erheblich aus Ausführungszeit. Wenn durch das Hinzufügen des Moduls zur *uses*-Anweisung eine zirkuläre Referenz entsteht, dann müssen Sie die Prozeduren zwischen den Modulen verschieben. Sie können aber auch ein neues Modul zum Projekt hinzufügen und die fragliche Prozedur dorthin auslagern.

Aufruf von *ExecMacro* aus einem Formular: Sie können alle Prozeduren ihres Projektes direkt aus dem Formular heraus aufrufen. Ersetzen Sie einen Aufruf wie

```
ExecMacro(MeinModul, MeineProzedur)
```

einfach durch

```
MeinModul.MeineProzedur
```

und prüfen Sie, ob MeinModul irgendwo im Projekt eingefügt ist.

4.6.4.3.25 ExecModal

Syntax

```
ExecModal(Kommentar: String [; Mode: Integer])
```

Kategorie

Oberflächenfunktion

Erklärung

ExecModal schaltet das aktuelle Datenfenster auf modal und zeigt den *Kommentar* im Panel an.

Mit dem optionalen Parameter Mode, können Sie das Panel ausschalten. Dazu setzen Sie Mode auf den Wert -1. Mit dem Vorgabewert 0 wird das Panel angezeigt.

Siehe auch

AppendRec, AppendRecs, EditRec, EditRecs, ViewRec, ViewRecs

4.6.4.3.26 FormularÖffnen

Syntax

```
FormularÖffnen(Formular: String)
OpenForm(Form: String)
```

Kategorie

Oberflächenfunktion

Erklärung

Öffnet ein Datenfenster für das angegebenen Formular.

Formular muss der vollständige Name des Formulars sein, also Tabellenname, Punkt, Formularname.

Beispiel

```
FormularÖffnen("KFZ.Eingabeformular");
```

Siehe auch

DatensätzeBearbeiten, Schließen

4.6.4.3.27 GibModus

Syntax

```
GibModus: Integer
GetMode: Integer
```

Kategorie

Oberflächenfunktion

Erklärung

GibModus ermittelt, auf welche Weise der aktuelle Datensatz im aktuellen Datenfenster gerade bearbeitet wird. Die möglichen Rückgabewerte sind:

< 0 -> kein Datenfenster aktiv

0 -> Datensatz wird betrachtet

1 -> Datensatz wird geändert

2 -> Datensatz wird neu eingegeben

Beispiel

```
..Ein Datenfenster öffnen
DatensätzeBearbeiten("KFZ.Fahrzeuge");
..Ein neues Datenfenster ist immer im Modus betrachten, daher zeigt die Box
'Modus: betrachten' an
Message('Modus: " + Choice(GetMode+1, 'betrachten', 'ändern', 'neu eingeben',
'Fehler'))
DatensätzeÄndern
..Jetzt wird der Text 'Modus: ändern' ausgegeben
Message('Modus: " + Choice(GetMode+1, 'betrachten', 'ändern', 'neu eingeben',
'Fehler'))
DatensätzeÄndern
..Nun sind wir wieder im Betrachten-Modus
Message('Modus: " + Choice(GetMode+1, 'betrachten', 'ändern', 'neu eingeben',
'Fehler'))
```

Siehe auch

NeueDatensätzeEingeben, DatensätzeÄndern

4.6.4.3.28 GibSicht

Syntax

```
GibSicht: Integer
GetView: Integer
```

Kategorie

Oberflächenfunktion

Erklärung

GibSicht ermittelt ob sich das aktuelle Datenfenster in der Tabellen- oder Formularsicht befindet.

Rückgabewerte:

1	Formularsicht
2	Tabellensicht

Beispiel

Die folgende Prozedur schaltet in die jeweils andere Sicht um:

```
PROCEDURE Sichtwechsel;
      IF GibSicht = 1
           SetzeSicht(2)
      ELSIF GibSicht = 2
           SetzeSicht(1)
      END
ENDPROC;
```

Siehe auch

SetzeSicht

4.6.4.3.29 Input

Syntax

```
Input(Prompt: String; Caption: String [; Options: Integer] [; var Text:
String]): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt ein Dialogfenster mit der Überschrift Caption, der Beschriftung *Prompt* und einem Eingabefeld an. Das Eingabefeld wird mit dem Inhalt des Parameters *Text* vorbelegt. Wenn das Dialogfenster mit OK geschlossen wird, steht der eingegebene Text anschließend wieder in *Text*.

Wenn Sie *Options* angeben und auf 1 setzen, wird die Eingabe verdeckt durchgeführt. Auf diese Weise können Sie ein Passwort-Abfrage realisieren.

Wenn Sie den Parameter *Text* weglassen, wird die globale Variable *T-Eingabe* verwendet. Dies wird allerdings nicht empfohlen.

Der Rückgabewert ist 1, falls die Eingabe bestätigt wurde, und 0 bei Abbruch.

Beispiel

```
vardef Stimmung: String;
Stimmung := "Super";
Input("Wie geht es Ihnen?", "Hallo", 0, Stimmung);
```

Siehe auch

ChooseFile, OrdnerAuswählen

4.6.4.3.30 Kopplung

Syntax

```
Kopplung
Coupling
```

Kategorie

Oberflächenfunktion

Erklärung

Falls das aktuelle Datenfeld im aktuellen Datenfenster ein Koppel- oder Relationsfeld ist, öffnet diese Funktion ein modales Datenfenster der angekoppelten Tabelle. In diesem Datenfenster können Datensätze ausgewählt werden, die mit dem aktuellen Datensatz des ursprünglichen Datenfensters verknüpft werden. Entspricht dem Menüpunkt *Bearbeiten/Kopplung* im Menü des Datenfensters.

Da *Kopplung* sich auf das aktuelle Feld bezieht, kann es nicht auf einen Schalter gelegt werden. Nur im Menü oder in den Makros *Beim Betreten* und *Beim Verlassen* macht es Sinn.

4.6.4.3.31 MaskenelementSuchen

Syntax

```
MaskenelementSuchen(Maskenelement: String): Control
FindControl(Maskelement: String): Control
```

Kategorie

Oberflächenfunktion

Erklärung

MaskenelementSuchen sucht ein Maskenelement in einem geöffneten Datenfenster. Die Funktion liefert eine Referenz auf ein Objekt vom Typ *Control*, bzw. den Wert *Null*, wenn kein Element mit dem übergebenen Namen vorhanden ist.

Mit dem Kommando *as* kann die Referenz auf einen speziellen *Control*-Typ wir z.B. *Edit* umgewandelt (typecast) werden. Dazu ist es unbedingt erforderlich, dass die strenge Typprüfung im Modul eingeschaltet ist (*SV 1*).

Beispiel

Diese Prozedur sucht im aktuellen Datenfenster nach dem Maskenelement *Spalte1Edit* und schreibt eine Zeichenkette in das Editierfeld.

```
procedure TextToEdit;
  vardef EditCtrl: Edit;
  EditCtrl := FindControl("Spalte1Edit") as Edit;
  if Assigned(EditCtrl)
    EditCtrl.Text := 'Das ist die Vorgabe';
  else
    Message('Spalte1Edit gibt es nicht');
  end;
endproc;
```

Siehe auch

Programmieren mit Objekten

4.6.4.3.32 Maximize/Maximieren

Syntax

```
Maximize
Maximimieren
```

Kategorie

Oberflächenfunktion

Erklärung

Maximiert das Formular. Solange das maximierte Formular geöffnet bleibt, werden alle Fenster innerhalb des Hauptfensters maximiert dargestellt, d.h. sie füllen das Hauptfenster vollständig aus.

Siehe auch

Minimize/Minimieren, Restore/Wiederherstellen

4.6.4.3.33 Message

Syntax

```
Message(Text: String; [Caption: String; Type: Integer]): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt eine Mitteilungsbox mit der Überschrift *Caption* und der Meldung *Text* an. Die Schalterkombination hängt von dem Wert von *Type* ab. Der Rückgabewert ist ein Code für den gedrückten Schalter.

Rückgabewerte:

1	OK-Schalter
2	Abbruch-Schalter
6	Ja-Schalter
7	Nein-Schalter

Beispiel

```
if Message("Wollen Sie den Text speichern?", "Schließen", 3) = 1
      ..Hier wird gespeichert
end
```

4.6.4.3.34 MessageType Aufzählung

Syntax

```
MessageType = (Ok, OkCancel, YesNo, YesNoCancel)
```

Kategorie

Benutzeroberfläche

Werte

Bezeichner	Wert	Bedeutung
Ok	1	Ok-Schalter
OkCancel	2	Ok- und Abbrechen-Schalter
YesNo	3	Ja- und Nein-Schalter
YesNoCancel	4	Ja-, Nein- und Abbrechen-Schalter

Beschreibung

Diese Aufzählung legt fest, welchen Typ eine Nachricht an den Benutzer hat.

Siehe auch

Message Prozedur

4.6.4.3.35 Minimieren/Minimize

Syntax

```
Minimize
Minimieren
```

Kategorie

Oberflächenfunktion

Erklärung

Minimiert das Formular, es wird dann als Ikone links unten im Hauptfenster dargestellt.

Siehe auch

Maximize/Maximieren, Restore/Wiederherstellen

4.6.4.3.36 NeueDatensätze

Syntax

```
NeueDatensätze(Kommentar: String)
AppendRecs(Comment: String)
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet das aktive Datenfenster auf modal, zeigt *Kommentar* im Panel an, sperrt alle Befehle außer den zur Neueingabe nötigen und fügt einen leeren Datensatz an. Der Benutzer kann beliebig viele neue Datensätze eingeben und bestätigen.

Beispiel

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
NeueDatensätze("Eingabe der neuen Fahrzeuge.");
Schließen
```

Siehe auch

NeuerDatensatz, DatensatzEditieren, DatensätzeEditieren, ExecModal, DatensätzeMarkieren, DatensatzBetrachten, DatensätzeBetrachten, NeueVerknüpfteDatensätzeEigeben, VerknüpfteDatensätze

4.6.4.3.37 NeueDatensätzeEingeben

Syntax

```
NeueDatensätzeEingeben
AppendNewRecords
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet im aktuellen Datenfenster den Modus für Neueingabe ein bzw. aus.

Entspricht dem Menüpunkt *Bearbeiten/Neue Datensätze* eingeben im Menü des Datenfensters.

Beispiel

```
..Ein Fenster öffnen und auf Neueingabe schalten
FormularÖffnen("KFZ.Fahrzeuge");
NeueDatensätzeEingeben
..Jetzt kann der Anwender neue Fahrzeuge erfassen
```

Siehe auch

GibModus , DatensätzeÄndern, DatensätzeBearbeiten, DatensatzLöschen

4.6.4.3.38 NeuerDatensatz

Syntax

```
NeuerDatensatz(Kommentar: String): Integer
AppendRec(Comment: String): Integer
```

Erklärung

Schaltet das aktive Datenfenster auf modal, zeigt *Kommentar* im Panel an, sperrt alle Befehle außer die zur Neueingabe nötigen und fügt einen leeren Datensatz an. Der Benutzer kann die Neueingabe mit OK oder Abbruch beenden. Der Rückgabewert ist die Satznummer des neuen Datensatzes oder 0, falls abgebrochen wurde.

Beispiel

```
DatensätzeBearbeiten("KFZ.Eingabeformular");
NeuerDatensatz("Geben Sie ein neues Fahrzeug ein.");
Message("Sie haben " + Bezeichnung + " eingegeben.");
Schließen
```

Siehe auch

NeueDatensätze, DatensatzAuswählen, DatensatzEditieren, DatensätzeEditieren,

DatensatzBetrachten, DatensätzeBetrachten, ExecModal, DatensätzeMarkieren, NeueVerknüpfteDatensätzeEingeben, VerknüpfteDatensätze

4.6.4.3.39 NeuenVerknüpftenDatensatzEingeben

Syntax

```
NeuenVerknüpftenDatensatzEingeben(Tabelle [,Kommentar: String] [, Formular: String]): Integer
AppendLinkedRec(Table [, Comment: String] [, Form: String]): Integer
```

Kategorie

Oberflächenfunktion

Beschreibung

Öffnet ein modales Datenfenster für die angegebenen Tabelle, um dort einen neuen Datensatz einzugeben, der automatisch mit dem aktuellen Datensatz des ursprünglichen Datenfensters verknüpft wird. *Kommentar* ist optional und wird im Panel angezeigt des modalen Fensters. Ebenfalls optional ist die Angabe des zu benutzenden Formulars. Falls dort nichts angegeben ist wird das erste Formular der Tabelle verwendet.

Beispiel

```
NeuenVerknüpftenDatensatzEingeben("KUNDEN", "Geben Sie einen weiteren Käufer ein.");
```

Siehe auch

NeueVerknüpfteDatensätzeEingeben VerknüpfteDatensätze, NeuerDatensatz, NeueDatensätze

4.6.4.3.40 NeueVerknüpfteDatensätzeEingeben

Syntax

```
NeueVerknüpfteDatensätzeEingeben(Tabellenname [,Kommentar: String] [, Formular: String])
AppendLinkedRecs(Table [, Comment: String] [, Form: String])
```

Kategorie

Oberflächenfunktion

Erklärung

Öffnet ein modales Datenfenster für die angegebenen Tabelle, um dort neue Datensätze einzugeben, die automatisch mit dem aktuellen Datensatz des ursprünglichen Datenfensters verknüpft werden. *Kommentar* ist optional und wird im Panel angezeigt.

Ebenfalls optional ist die Angabe des zu benutzenden Formulars. Falls dort nichts angegeben ist wird das erste Formular der Tabelle verwendet.

Entspricht dem Menüpunkt *Bearbeiten/Neue verknüpfte Datensätze eingeben* im Menü des Datenfensters.

Beispiel

Eingabe weiterer Käufer des aktuellen Fahrzeuges im Datenfenster von KFZ:

```
NeueVerknüpfteDatensätzeEingeben("KUNDEN", "Geben Sie weitere Käufer ein.");
```

Siehe auch

NeuenVerknüpftenDatensatzEingeben, VerknüpfteDatensätze, NeuerDatensatz, NeueDatensätze

4.6.4.3.41 Restore/Wiederherstellen

Syntax

```
Restore
Wiederherstellen
```

Kategorie

Oberflächenfunktion

Erklärung

Stellt die ursprüngliche Formulargröße wieder her, wenn es minimiert oder maximiert ist.

Siehe auch

Minimize/Minimieren, Maximize/Maximieren

4.6.4.3.42 Schließen

Syntax

```
Schließen
CloseWnd
```

Kategorie

Oberflächenfunktion

Erklärung

Schließt das aktuelle Fenster

Schließen kann als eigenständiger Befehl auf einen Schalter gelegt werden, um das zugehörige Datenfenster zu schließen. Eine andere Möglichkeit ist in Zusammenhang mit *FormularÖffnen* und z.B. *DatensatzEditieren*, um eine gesteuerte Eingabe für den Benutzer zu realisieren.

Wird ein Formular mittels *ExecDialog* als modaler Dialog ausgeführt, erhält der im Formular enthaltene Ok-Knopf *CloseWnd* als auszuführendes Makro.

Beispiel

```
OpenForm("KUNDEN.Kundenkartei");
ShowRec(-1)
EditRec("Sie können jetzt den ersten Datensatz der Tabelle bearbeiten");
CloseWnd
```

Siehe auch

FormularÖffnen, DatensätzeBearbeiten, DatensatzEditieren, ExecDialog, Abbruch

4.6.4.3.43 SetzeSicht

Syntax

```
SetzeSicht(Modus: Integer))
SetView(Mode: Integer);
```

Kategorie

Oberflächenfunktion

Erklärung

Schaltet das aktuelle Datenfenster zwischen Tabellen- und Formularsicht um. Die Werte für den Parameter *Modus* sind:

0	Umschalten
1	Auf Formularsicht schalten
2	Auf Tabellensicht schalten

Beispiel

Schaltet das aktive Datenfenster in die Tabellensicht

```
SetzeSicht(2);
```

Siehe auch

GibSicht

4.6.4.3.44 SetzeTabZiel

Syntax

```
SetzeTabZiel(Vorwärts, Rückwärts: Real [, SeitennrVorwärts, SeitennrRückwärts:
Integer])
SetTabTarget(Forward, Backward: Real[, PageNoForward, PageNoBackward:
Integer])
```

Kategorie

Oberflächenfunktion

Erklärung

Mit *SetzeTabZiel* können Sie die normale Reihenfolge der Felder im Formular vorübergehend abändern. Drückt der Anwender in einem Eingabefeld die Tabulator oder die Eingabe-Taste, so wird standardmäßig das nächste Feld fokussiert. Dabei gilt die Reihenfolge, wie sie im Formular-Editor unter Nummerierung festgelegt wurde.

Mit *SetTabTarget* oder *SetzeTabZiel* können Sie dieses Standard-Verhalten ändern und selbst das nächste zu fokussierende Feld angeben:

Vorwärts Die Nummer des Feldes, das mit der Tabulator-Taste der Pfeiltaste unten oder der Eingabe-Taste fokussiert wird.

Rückwärts Die Nummer des Feldes, das mit Umschalt-Tabulator oder der Pfeiltaste oben fokussiert wird.

Für mehrseitige Formulare gilt zusätzlich:

SeitennrVorwärts Die Formularseite, die zusammen mit Vorwärts angezeigt werden soll.

SeitennrRückwärts Die Formularseite, die zusammen mit Rückwärts angezeigt werden soll.

Zum Wiederherstellen des Standard-Verhaltens setzen Sie alle Werte auf 0.

Die Verwendung erfolgt innerhalb von Formularprozeduren, bzw. Makros , meistens im Ereignis *BeimVerlassen*.

Beispiel

Ein Formular enthält ein Ja/Nein-Feld mit der Nummer 1, indem der Anwender festlegt, ob ein Rechnungs-Posten mit MWSt. versehen wird. Falls ja, soll das Feld 2 als nächstes fokussiert werde, weil dort der MWSt.-Satz einzutragen ist. Falls nein, kann man gleich mit Feld 3 weitermachen. In den Eigenschaften unter *BeimVerlassen* des Ja/Nein-Feldes wird folgendes Makro eingetragen:

```
SetzeTabZiel(Choice(MWSt-pflichtig, 2, 3), 0)
```

Siehe auch

SeiteAnzeigen

4.6.4.3.45 SeiteAnzeigen

Syntax

```
SeiteAnzeigen(SeitenNr: Integer): Integer
ViewPage(PageNo: Integer): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

SeiteAnzeigen schaltet auf die Formularseite mit der Nummer *SeiteNr* und gibt die Nummer der Formularseite zurück.

Einige spezielle Werte für *SeiteNr*:

-4	Schaltet auf die erste Seite des Formulars
-3	Schaltet eine Formularseite zurück
-2	Schaltet eine Formularseite vorwärts
-1	Schaltet auf die letzte Formularseite
0	Schaltet nichts um

Beispiel

```
T-Eingabe := "0";
IF Input("Welche Formularseite wollen Sie sehen?", "Umschalten") = 1
      SeiteAnzeigen(Val(T-Eingabe))
END
```

Siehe auch

AnDenAnfangBlättern, AnDasEndeBlättern, Weiterblättern, Zurückblättern

4.6.4.3.46 StarteDialog

Syntax

```
ExecDialog(Form: String [; RecordNo: Integer]): Integer
StarteDialog(Formular: String [; SatzNummer: Integer]): Integer
```

Kategorie

Oberflächenfunktion

Erklärung

Die Funktion öffnet das Formular *Form* modal als Dialog. Es wird der erste Datensatz der Tabelle bearbeitet, außer es ist ein anderer gültiger Datensatz im optionalen Parameter *RecordNo/ Satznummer* angegeben. Das Formular erscheint in seiner Originalgröße am Bildschirm, auch wenn die anderen Fenster auf Vollbild eingestellt sind, und es gibt keine Schalterleiste und kein Menü. Wenn Sie beispielsweise eine SYSTEM-Tabelle anlegen, in der verschiedene globale Variablen Ihrer Anwendung abgelegt sind, können Sie auf diese Weise sehr komfortable Dialoge programmieren. Dialog-Formulare sollten einen OK-Button (Makro: Schließen) und einen Cancel-Button (Makro: Abbruch) haben.

Rückgabewerte:

1 OK-Schalter

2 Abbruch-Schalter

Hinweis

Die Vorbelegungen in der Eingabekontrolle des Dialog-Formulares sind in Verbindung mit *StarteDialog* leider nicht anwendbar. Diese werden ja nur aktiviert, wenn ein neuer Datensatz angelegt wird, was hier meistens nicht der Fall ist. Vorbelegungen sind daher, wie im nachfolgenden Beispiel gezeigt, direkt in die entsprechenden Felder des ersten Datensatzes der Dialog-Tabelle zu schreiben.

Beispiel

```
ReadRec(SYSTEM, 1)
SYSTEM.IntegerEingabe := 0
WriteRec(SYSTEM, 1)
IF StarteDialog("SYSTEM.IntegerEingabe") = 1
      Message("Der eingegebene Integer-Wert ist: " +
Str(SYSTEM.IntegerEingabe))
ELSE
      Message("Der Dialog wurde mit Abbruch beendet")
END
```

Siehe auch

Abbruch, Schließen

4.6.4.3.47 Sortierung

Syntax

```
Sortierung(IndexName: String; Modus: Integer)
SetSortOrder(IndexName: String; Mode: Integer)
```

Kategorie

Oberflächenfunktion

Erklärung

Sortierung setzt die Reihenfolge der Datensätze im aktuellen Datenfenster. *IndexName* muss der Name eines vorhandenen Index sein, dessen Sortierordnung verwendet werden soll. Auch *Nummer* und *Markierung* ist möglich. Der zweite Parameter ist optional, die Voreinstellung ist Modus = 0.

Werte für Modus:

0 Alle Datensätze anzeigen
1 Aktuelle Auswahl verwenden (Nur markierte oder alle Datensätze anzeigen)
2 Nur markierte Datensätze anzeigen

Beispiel

Tabelle KUNDEN im KFZ-Projekt hat einen Index KUNDEN.ID. Das Datenfenster von KUNDEN ist aktiv:

```
Sortierung("KUNDEN.ID");
Message("Jetzt sehen Sie die Kunden alphabetisch geordnet.");
Sortierung("Nummer");
Message("Jetzt sehen Sie die Kunden in der Reihenfolge der Eingabe");
```

Siehe auch

Access, IndName, IndNo, SetAccess

4.6.4.3.48 Verknüpfte Datensätze

Syntax

```
VerknüpfteDatensätze(Tabelle: String; Ansicht: Integer; [Kommentar, Formular: String])
ViewLinkedRecs(Table: String; View: Integer; [Comment, Form: String])
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt ein Datenfenster mit den verknüpften Datensätze aus der angegebenen Tabelle an. Entspricht dem Menüpunkt *Ansicht/Verknüpfte Datensätze*. Die Datensätze werden je nach dem Wert von *Ansicht* in Tabellensicht oder in Formularsicht angezeigt:

1 Formularsicht

2 Tabellensicht.

Der *Kommentar* erscheint im Panel der Formularsicht.

Wenn Ansicht nicht angegeben wird, werden die Daten in der Tabellensicht dargestellt. Falls *Kommentar* nicht angegeben wird, erscheint der Standard-Kommentar.

Mit dem optionalen Parameter *Formular* können Sie angeben, welches Formular zur Anzeige verwendet werden soll. Falls dort nichts angegeben ist wird das erste Formular der Tabelle benutzt.

Beispiel

Anzeige aller Käufer des aktuellen Fahrzeuges im KFZ-Datenfenster:

```
VerknüpfteDatensätze("KUNDEN", 2, "Hier die Käufer des Fahrzeuges " +
Bezeichnung);
```

Siehe auch

NeueVerknüpfteDatensätzeEingeben, NeuerDatensatz, NeueDatensätze, DatensatzAuswählen, DatensatzEditieren, DatensätzeEditieren, DatensätzeMarkieren, DatensatzBetrachten, DatensätzeBetrachten

4.6.4.3.49 WartenStart

Syntax

```
WartenStart(Meldung: String)
ShowWait(Message: String)
```

Kategorie

Oberflächenfunktion

Erklärung

WartenStart zeigt ein Dialogfenster mit der Meldung an. Das Dialogfenster bleibt bis zum nächsten Aufruf von *WartenStop* auf dem Bildschirm. Solange das Dialogfenster angezeigt ist, kann der Anwender keine Aktionen mit *TurboDB Studio* ausführen.

WartenStart kann auch mehrmals hintereinander aufgerufen werden, um den angezeigten Text zu ändern.

Beispiel

```
Procedure LangeOperationen
```

```
        WartenStart("Jetzt wird eine längerdauernde Berechnung durchgeführt...")
        ..Hier kommen die Anweisungen für die Berechnung
        WartenStart("Jetzt wird die Sortierung eingestellt...")
        ..Hier kommen die Anweisungen für die Sortierung
        WartenStop
  EndProc
```

Siehe auch

WartenStop, Message

4.6.4.3.50 WartenStop

Syntax

```
WartenStop
HideWait
```

Kategorie

Oberflächenfunktion

Erklärung

Wenn der Warte-Dialog angezeigt ist, wird er von *WartenStop* vom Bildschirm genommen.

Beispiel

siehe WartenStart

4.6.4.3.51 Weiterblättern

Syntax

```
Weiterblättern
ViewNextPage
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktuellen Datenfenster die nächste Seite eines mehrseitigen Formulars an.

Entspricht dem Menüpunkt *Ansicht/Seite/Weiterblättern* im Menü des Datenfensters.

Beispiel

In einem mehrseitigen Formular können Makroschalter angebracht werden um das Umschalten zwischen den einzelnen Seiten zu ermöglichen. Der Schalter mit der Aufschrift "Nächste Seite" erhält als Makro den Aufruf von *Weiterblättern*.

Siehe auch

AnDenAnfangBlättern, AnDasEndeBlättern, Zurückblättern, SeiteAnzeigen

4.6.4.3.52 Zurückblättern

Syntax

```
Zurückblättern
ViewPrevPage
```

Kategorie

Oberflächenfunktion

Erklärung

Zeigt im aktuellen Datenfenster die vorherige Seite eines mehrseitigen Formulars an.

Entspricht dem Menüpunkt *Ansicht/Seite/Zurückblättern* im Menü des Datenfensters.

Beispiel

In einem mehrseitigen Formular können Makroschalter angebracht werden um das Umschalten zwischen den einzelnen Seiten zu ermöglichen. Der Schalter mit der Aufschrift "Vorherige Seite" erhält als Makro den Aufruf von *Zurückblättern*.

Siehe auch

AnDenAnfangBlättern, AnDasEndeBlättern, Weiterblättern

4.6.5 Klasse Steuerelement

4.6.5.1 Control-Objekt

Mit dem Control-Objekt kann auf die Steuerelemente im Formular zugegriffen werden. Einen Verweis auf ein Control-Objekt erhalten Sie entweder dynamisch mit der Funktion *FindControl* oder statisch über den Namen des Steuerelements in einem Formular-Modul.

4.6.5.2 Color

Syntax

```
Color: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest den aktuellen Farbwert für das Steuerlement.

Beispiel

Setzt die Farbe des Steuerelements auf violett:

```
TheControl.Color := RGB(255, 0, 255);
```

4.6.5.3 Height

Syntax

```
Height: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die aktuelle Höhe des Steuerelements.

4.6.5.4 Hint

Syntax

```
Hint: string;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die Eigenschaft Hint des Steuerelements.

4.6.5.5 Left

Syntax

```
Left: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die aktuellen horizontale Position des Steuerelements.

4.6.5.6 Top

Syntax

```
Top: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die aktuelle vertikale Position des Steuerelements.

4.6.5.7 Visible

Syntax

```
Visible: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest den aktuellen Wert für die Sichtbarkeit des Steuerelements. Dabei steht 0 für unsichtbar und 1 für sichtbar.

Beispiel

Diese Anweisung in einem Formular-Modul schaltet das Steuerlement *TheControl* zwischen sichtbar und unsichtbar um.

```
TheControl.Visible := 1 - TheControl.Visible;
```

4.6.5.8 Width

Syntax

```
Width: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die aktuelle Breite des Steuerelements.

4.6.5.9 Text

Syntax

```
Text: String;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest den aktuellen Text des Steuerelements. Diese Eigenschaft ist für folgende Steuerelement-Typen *Label*, *Reference*, *Edit*, *DateTimePicker*, *CheckBox*, *ComboBox*, *Frame*, *TabSheet*, *Button*, *Memo*, *RadioGroup*, *LookupTable* und *MultiLinkEdit* verfügbar.

Beispiel

Siehe FindControl

4.6.5.10 Enabled

Syntax

```
Enabled: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die Eigenschaft *Enabled* des Steuerelements. Diese Eigenschaft ist nur für das Steuerelement *Timer* verfügbar.

0 Ausgeschaltet
1 Eingeschaltet

4.6.5.11 Interval

Syntax

```
Interval: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die Eigenschaft *Interval* des Steuerelements. Diese Eigenschaft ist nur für das Steuerelement *Timer* verfügbar.

Die Angabe erfolgt in 1/10 Sekunden und legt fest in welchen Abständen das Zeitgeber-Element das unter *Aktion* eingetragene Makro ausführt.

4.6.5.12 Checked

Syntax

```
Checked: Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die Eigenschaft *Checked* des Steuerelements. Diese Eigenschaft ist nur für das Steuerelement *CheckBox* verfügbar.

0 Ausgeschaltet
1 Eingeschaltet

4.6.5.13 Formula

Syntax

```
Formula: String;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die Eigenschaft *Formula* des Steuerelements. Diese Eigenschaft ist nur für das Steuerelement *FormulaLabel* verfügbar.

4.6.5.14 Reference

Syntax

```
Reference: String;
```

Kategorie

Oberflächenfunktion

Erklärung

Setzt oder liest die Eigenschaft *Reference* des Steuerelements. Diese Eigenschaft ist nur für das Steuerelement *ReferenceLabel* verfügbar.

4.6.5.15 ViewPage

Syntax

```
procedure ViewPage(PageIdx: Integer): Integer;
```

Kategorie

Oberflächenfunktion

Erklärung

Führt die Methode *ViewPage* des Steuerelements aus. Diese Eigenschaft ist nur für das Steuerelement *PageControl* verfügbar.

PageControl.ViewPage funktioniert analog zu *SeiteAnzeigen/ViewPage* des Datenfensters.

4.6.5.16 FileName

Syntax

```
FileName: String;
```

Kategorie

Oberflächenfunktion

Erklärung

Diese Eigenschaft ist nur für das Steuerelement *Blob* verfügbar. Setzt oder liest den Pfad zu einer Bild-Datei.

Ist das Steuerelement beim Setzen der Eigenschaft mit einem Datenfeld verbunden, wird diese Verbindung aufgehoben.

4.6.6 Druck-Funktionen

4.6.6.1 Druck-Funktionen

Mit diesen Befehlen starten Sie Ausdrucke und legen das Druckziel fest.

4.6.6.2 Drucken

Syntax

```
Drucken(Elementname: String [; Modus: Integer])
Run(Elementname: String [; Mode: Integer])
```

Erklärung

Das Projektelement mit dem vollständigen Namen *Elementname* wird gedruckt. Wenn Modus den Wert 0 hat, entspricht Run dem Menüpunkt *Ausführen/Drucken* im Menü des Datenfensters und ist identisch mit der Menü-Funktion *Drucken*. D.h. vor dem Ausdruck wird ein Druckdialog mit dem Druckziel angezeigt und kann abgebrochen werden. Ist *Modus* ungleich 0, wird ohne Dialog sofort gedruckt. Die Wert von *Modus* im einzelnen:

0	Drucken mit Druckdialog (Vorgabe)
1	Drucken auf eingestellten Drucker
2	Drucken in Druckvorschau
3	Drucken in Datei mit den eingestellten Formatangaben

Der Drucker und das Papierformat können vor dem Druck mit *SetzeDrucker* eingestellt werden.

Falls ein Ausdruck in eine Datei erfolgen soll, kann mit *SetzeAusgabeDatei* die Zieldatei festgelegt werden.

Beispiel

Die Prozedur druckt den Datenbankjob Anschreiben für den aktuellen Datensatz der Tabelle KUNDEN. Voraussetzung ist, dass im Datenbankjob *Anschreiben* der Zugriff auf *Markierung* gesetzt ist.

```
PROCEDURE DruckeAktuellenDatensatz;
  RemoveAllStars(1);
  PutStar;
  Drucken('KUNDEN.Anschreiben');
```

```
ENDPROC;
```

Siehe auch

DruckerEinrichten, SetzeAusgabeDatei

4.6.6.3 GibAktuellenDrucker

Syntax

```
GibAktuellenDrucker: String
GetCurrentPrinter: String
```

Erklärung

Ermittelt den aktuell eingestellten Druckernamen

Beispiel

```
Message("Der aktuelle Drucker ist " + GibAktuellenDrucker);
```

Siehe auch

SetPrinter

4.6.6.4 OpenReport

Syntax

```
OpenReport(Berichtname: String)
```

Erklärung

Bereitet einen Bericht zum anschließenden Drucken mit *PrintReport* vor. Der Sinn dieser Funktion ist, dass man zwischen *OpenReport* und *PrintDocument* Einstellungen an den Tabellen vornehmen kann (wie Sortierung und Auswahl), die dann beim Drucken beachtet werden. In den Bericht-Eigenschaften muss dazu die Eigenschaft *Vorhandene Sortierung verwenden* selektiert sein.

Beispiel

Siehe PrintDocument

Siehe auch

PrintDocument

4.6.6.5 Print

Syntax

```
Print(Ausgabeformat)
```

Erklärung

Das Ausgabeformat wird ausgewertet und auf dem aktuellen Ausgabegerät ausgegeben. Print erhöht den internen Zeilenzähler, so dass ein automatischer Seitenumbruch möglich ist. Zeilen müssen mit Schrägstrich abgeschlossen werden, damit ein Zeilenumbruch erfolgt. In Modulen gibt es keine Ausgabemöglichkeit auf dem Bildschirm.

Beispiel

```
Print(Name:20 Vorname:20. Telefon/);
```

4.6.6.6 PrintDocument

Syntax

```
PrintDocument(Modus: Integer)
```

Erklärung

Druckt den mit *OpenReport* erstellten Bericht aus. Es ist damit möglich auch ohne geöffnetes Datenfenster eine selektierte Datenmenge auszugeben. Das Ausgabeziel kann mit dem Parameter *Modus* angegeben werden.

Die Wert von *Modus* im einzelnen:

0 Drucken mit Druckdialog (Vorgabe)

1 Drucken auf eingestellten Drucker
2 Drucken in Druckvorschau
3 Drucken in Datei mit den eingestellten Formatangaben

Der Drucker und das Papierformat können vor dem Druck mit *SetzeDrucker* eingestellt werden.

Beispiel

Die Prozedur druckt ein Anschreiben für alle Kunden, die einen Vertreterbesuch gewünscht haben. Stellen Sie sicher, dass bei den Bericht-Eigenschaften der Eintrag *Vorhandene Sortierung verwenden* selektiert ist.

```
PROCEDURE DruckeTerminliste;
   OpenReport("KUNDEN.Vertreterbesuch")
   Link(KUNDEN,Besuch=JA, SetMark(KUNDEN, RecNo(KUNDEN)))
   Access(KUNDEN, "Markierung")
   PrintDocument(0)
ENDPROC;
```

Siehe auch

OpenReport

4.6.6.7 SetzeAusgabeDatei

Syntax

```
SetzeAusgabeDatei(DateiName: String; DateiFormat: Integer)
SetOutputFile(FileName: String; FileFormat: Integer)
```

Erklärung

Falls mit der Funktion *Drucken* (*Run*) ein Ausdruck in eine Datei erfolgen soll, kann mit *SetzeAusgabeDatei* die Zieldatei festgelegt werden.

Es kann zwischen mehreren Dateiformaten gewählt werden:

1 Druckerdaten
2 DOS-Text
3 Windows-Text
4 HTML

Beispiel

```
procedure PrintReport;
   if Input("Zieldatei") = 1
     SetzeAusgabeDatei(T-Eingabe, 4)
     Drucken("KUNDEN.Liste", 3);
   end;
endproc;
```

Siehe auch

Drucken, DruckerEinrichten

4.6.6.8 SetzeDrucker

Syntax

```
SetzeDrucker(DruckerName: String[; Format, PapierGröße, Breite, Länge:
Integer])
SetPrinter(PrinterName: String[; Format, Paper, Width, Length: Integer])
```

Erklärung

Stellt den verwendeten Drucker ein. Alle Parameter außer *DruckerName* sind optional.

Druckername Wählt einen Drucker unter den in Windows installierten Druckern aus. Dieser Name muss genau mit dem Namen übereinstimmen, der auch bei *Datei/Drucker einrichten* angezeigt wird. Geben Sie einen Leerstring an, um den eingestellten Drucker zu belassen. Netzwerkdrucker haben eine eigene Schreibweise: \\<*Server-Name*>*Druckername* also z.B. \\ELLI2\HP4L. Die Druckernamen sind in der *WIN.INI* (Windows 3.1 und Windows 95) bzw. in der Registry *HKEY_CURRENT_USER/Software/Microsoft/WindowsNT/CurrentVersion/Devices* zu finden.

Format	Wählt zwischen Hochformat (*Format* = 1) und Querformat (*Format* = 2). Wenn Sie 0 angeben, wird das eingestellte Format belassen. (Optional)
PapierGröße	Wählt aus den vorhandenen Papiersorten. Eine Liste dieser Papiersorten finden Sie weiter unten. Wenn Sie *PapierGröße* auf 0 setzen, bleibt die vorhandene Papiergröße unverändert außer Sie geben zusätzlich noch bei Breite oder Länge etwas an. Die Einstellung der Papiergröße bleibt unverändert, wenn der Drucker die Papiersorte nicht unterstützt. (Optional)
Breite	Gibt die Papierbreite in Zehntel Millimeter an. Wird nur berücksichtigt, wenn *PapierGröße* 0 ist. (Optional)
Höhe	Gibt die Papierlänge in Zehntel Millimeter an. Wird nur berücksichtigt, wenn *PapierGröße* 0 ist. (Optional)

Papiersorten

1	Letter 8 1/2 x 11 in
2	Letter Small 8 1/2 x 11 in
6	Statement 5 1/2 x 8 1/2 in
7	Executive 7 1/4 x 10 1/2 in
5	Legal 8 1/2 x 14 in
8	A3 297 x 420 mm
9	A4 210 x 297 mm
10	A4 Small 210 x 297 mm
11	A5 148 x 210 mm
12	B4 250 x 354 mm
13	B5 182 x 257 mm
20	Envelope #10 4 1/8 x 9 1/2
27	Envelope DL 110 x 220mm
28	Envelope C5 162 x 229 mm
29	Envelope C3 324 x 458 mm
30	C4 229 x 324 mm
33	Envelope B4 250 x 353 mm
34	Envelope B5 176 x 250 mm
37	Envelope Monarch 3.875 x 7.5 in
39	US Std Fanfold 14 7/8 x 11 in
40	German Std Fanfold 8 1/2 x 12 in
41	German Legal Fanfold 8 1/2 x 13 in

Beispiele

Zum Ausdruck einer Farbseite wird der Drucker vorübergehend gewechselt:

```
SetzeDrucker("Epson Stylus COLOR");
Run("VIDEO.Farbdruck", 1);
SetzeDrucker("HP LaserJet 4L");
```

Es wird ein Bericht im Querformat gedruckt:

```
SetzeDrucker("", 2);
Run("VIDEO.BreiteListe", 1);
SetzeDrucker("", 1);
```

Umstellen des Drucker zum Bedrucken eines Umschlags:

```
SetzeDrucker("", 0, 34);
```

Umstellen auf ein nicht standardisiertes Papierformat 20 cm x 20 cm:

```
SetzeDrucker("", 0, 0, 2000, 2000);
```

Siehe auch

Drucken, GibAktuellenDrucker

4.7 Datenaustausch und Multimedia

4.7.1 Multimedia

4.7.1.1 Multimedia

Mit den Multimedia-Funktionen können Sie Bilder, Klänge und Filme speichern, verarbeiten und abspielen.

4.7.1.2 MediumPause

Syntax

```
MediumPause
PauseMedia
```

Erklärung

Hält das Abspielen der aktiven Media-Datei an. Bei erneutem Aufruf der Funktion wird das Abspielen fortgesetzt.

Wurde keine Media-Datei mittels *PlayMedia* gestartet, bleibt der Aufruf der Funktion wirkungslos.

Beispiel

Siehe SpieleMedium

Siehe auch

SpieleMedium, MediaStop

4.7.1.3 MediumSpielen

Syntax

```
MediumSpielen(Dateiname: String)
PlayMedia(Filename: String)
```

Erklärung

Dateiname ist der Pfad zu einer beliebigen Multimedia-Datei, für die Ihr Rechner ausgerüstet ist. Das kann ein Klang (*.WAV), eine MIDI-Datei (*.MID), ein Video (*.AVI) oder eine andere Musik- oder Film-Datei sein. *SpieleMedium* öffnet die Datei und beginnt mit dem Abspielen. Ihre Anwendung läuft sofort nach dem Start des Abspielens weiter. Die Musik oder der Film laufen also im Hintergrund.

Beispiel

Angenommen, in der Tabelle SONGS befindet sich das Datenfeld Klangdatei, welches den Dateinamen für das Stück enthält. Dann können Sie in Ihr Formular drei Schalter einbauen, mit denen das Stück abgespielt werden kann.

Schalter 1 mit der Aufschrift "Spielen"

```
MediumSpielen(Klangdatei)
```

Schalter 2 mit der Aufschrift "Stop"

```
MediumStop
```

Schalter 3 mit der Aufschrift "Pause"

```
MediumPause
```

Siehe auch

PlaySound, MediumStop, MediumPause

4.7.1.4 MediumStop

Syntax

```
MediumStop
StopMedia
```

Erklärung

Beendet das Abspielen der aktiven Mediadatei.

Wurde keine Media-Datei mittels *MediumSpielen* gestartet, bleibt der Aufruf der Funktion wirkungslos.

Beispiel

siehe MediumSpielen

Siehe auch

MediumSpielen, MediumPause

4.7.1.5 PlaySound

Syntax

```
PlaySound(Blob: Spalte): Integer
```

Erklärung

Spielt den im angegebenen Feld abgelegten Klang ab. Das Feld muss ein BLOB-Feld sein und auch wirklich einen Klang enthalten. *PlaySound* kann nur kurze Klang-Dateien spielen und wartet, bis das Abspielen beendet ist. Abbruch ist nicht möglich. Ihr System muss über eine Soundkarte oder zumindest über einen Treiber für den eingebauten Lautsprecher verfügen.

Verwenden Sie *PlaySound* zum Abspielen kurzer Geräusche wie sie z.B. auch in der Windows-Systemsteuerung eingestellt werden können. Für längere Klänge wie z.B. Musik-Stücke ist die Funktion *PlayMedia* besser geeignet.

Rückgabewerte

0	Klang wurde erfolgreich abgespielt oder das Blob ist leer.
-1	Das Feld ist kein Blob, das Blob hat den falschen Typ oder es kann nicht gelesen werden.
-2	Beim Abspielen ist ein Fehler aufgetreten (z.B. hat der Computer keine Soundkarte)

Beispiel

Hier wird angenommen, dass in der aktuellen Primärtabelle ein Bild-Feld namens *Gezwitscher* besteht.

```
PlaySound(Gezwitscher)
```

Siehe auch

PlayMedia

4.7.2 Schnittstellen

4.7.2.1 Schnittstellen

Funktionen zur Verwaltung der seriellen Schnittstelle.

4.7.2.2 CommClose

Syntax

```
CommClose(Kanalnummer: Integer): Integer
```

Erklärung

Schließt den Kommunikationskanal mit der übergebenen Kanalnummer. Liefert Null oder einen negativen Fehlercode.

Siehe auch

CommOpen CommIn, CommOut, CommMode, CommRead, CommWrite, CommMode, CommState

4.7.2.3 CommIn

Syntax

```
CommIn(Kanalnummer: Integer): Integer
```

Erklärung

Liefert die Anzahl der Zeichen im Lesepuffer oder, falls das Ergebnis negativ ist, den Fehlercode.

Siehe auch

CommClose, CommOpen, CommOut, CommMode, CommRead, CommWrite, CommMode

4.7.2.4 CommMode

Syntax

```
CommMode(Kanalnummer, Baudrate, Bits, Parity, Stopbits, Timeout: Integer) :
Integer
```

Erklärung

Mit dieser Funktion wird die Schnittstelle initialisiert. Die Initialisierung ist nicht notwendig, wenn die Voreinstellung von Windows benützt wird. Die folgenden Werte sind erlaubt:

Baudrate	entsprechend IO-Port (meist maximal 19200), zwingend erforderlich
Bits	7 oder 8, zwingend erforderlich
Parity	0 (keine), 1 (ungerade), 2 (gerade), optional
Stopbits	1 (1 Stopbit), 2 (1,5 Stopbits), 3 (2 Stopbits), optional
Timeout	Wartezeit in Millsekunden, optional

Optionale Parameter können von rechts her weggelassen werden.

Das Ergebnis ist 0, wenn die Initialisierung geklappt hat, andernfalls -1.

Siehe auch

CommClose, CommIn, CommOpen, CommOut, CommRead, CommWrite, CommMode, CommState

4.7.2.5 CommOpen

Syntax

```
CommOpen(Gerät : String; [LesePuffer, SchreibPuffer : Integer]) : Integer
```

Erklärung

Öffnet eine Schnittstelle und liefert eine Kanalnummer für das gewünschte Gerät (z.B. "COM2") zurück. Über diese Kanalnummer sprechen Sie die Schnittstelle in folgenden Befehlen an. *LesePuffer* und *SchreibPuffer* geben die gewünschte Größe der Puffer in der Schnittstelle an. Der Vorgabewert ist hier jeweils 1.

Wenn der Rückgabewert kleiner als 0 ist, ist ein Fehler aufgetreten. Hier eine Liste der möglichen Fehlercodes:

Code	Bedeutung
-1	Ungültige oder nicht unterstützte ID
-2	Gerät bereits geöffnet
-3	Gerät nicht bereit
-4	Es können keine Warteschlangen eingerichtet werden (zuwenig Speicher)
-5	Fehler in Standard-Parametern
-10	Hardware nicht vorhanden (wird von einem anderen Gerät gesperrt)
-11	Ungültige Bytegröße
-12	Nicht unterstützte Baudrate

Siehe auch

CommClose, CommIn, CommOut, CommMode, CommRead, CommWrite, CommMode, CommState

4.7.2.6 CommOut

Syntax

```
CommOut(Kanalnummer: Integer): Integer
```

Erklärung

Liefert die Anzahl der Zeichen im Schreibpuffer oder, falls das Ergebnis negativ ist, den Fehlercode.

Siehe auch

CommClose, CommIn, , CommMode, CommOpen, CommRead, CommWrite, CommMode, CommState

4.7.2.7 CommRead

Syntax

```
CommRead(Kanalnummer: Integer; AnzahlZeichen: Integer): String
```

Erklärung

Liest *AnzahlZeichen* Zeichen aus dem Kommunikationskanal. Mit der Funktion *CommIn* kann vor dem Lesen geprüft werden, ob bereits genügend Zeichen empfangen wurden. Die gelesenen Zeichen werden als String zurückgegeben.

Siehe auch

CommClose, CommIn, CommOut, CommMode, CommOpen, CommWrite, CommMode, CommState

4.7.2.8 CommState

Syntax

```
CommState(Kanalnummer: Integer): Integer
```

Erklärung

Liefert das Status-Flag des Kommunikationskanals. Dieses Flag ist bitweise zu interpretieren:

Bit-Nr	Bit-Wert	Bedeutung
0	1	Bestimmt, ob die Übertragung darauf wartet, dass das CTS-Signal (clear-to-send) gesendet wird
1	2	Bestimmt, ob die Übertragung darauf wartet, dass das DSR-Signal (data-set-ready) gesendet wird
2	4	Bestimmt, ob die Übertragung darauf wartet, dass das RLSD-Signal (receive-line-signal-detect) gesendet wird
3	8	Bestimmt, ob die Übertragung wartet, wenn das Zeichen XOFF empfangen wurde
4	16	Bestimmt, ob die Übertragung wartet, wenn das Zeichen XOFF übertragen wurde. Die Übertragung muss nach dem Senden eines XOFF-Zeichens für Systeme gestoppt werden, die das folgende Zeichen unabhängig von seinem tatsächlichen Wert als XON interpretieren
5	32	Bestimmt, ob das Zeichen für Dateiende (EOF) empfangen worden ist
6	64	Bestimmt, ob ein Zeichen darauf wartet, übertragen zu werden

Beispiel

siehe TestBit

Siehe auch

CommOpen, CommClose, CommIn, CommOut, CommMode, CommRead, CommWrite, CommMode, TestBit

4.7.2.9 CommWrite

Syntax

```
CommWrite(CommHandle: Integer; Ausgabe: String): Integer
```

Erklärung

Schreibt den übergebenen String in den Kommunikationskanal. Es ist darauf zu achten, dass die Länge des Strings den bei *CommOpen* angegeben Schreibpuffer nicht übersteigt, da ansonsten die Ausgabe verstümmelt wird. Mit *CommOut* kann vor der Ausgabe geprüft werden, ob der Ausgabepuffer leer ist. Das Ergebnis enthält die Anzahl der gesendete Zeichen, wenn die Übertragung erfolgreich durchgeführt wurde, andernfalls wird der negative Fehlercode geliefert.

Siehe auch

CommOpen CommIn, CommOut, CommMode, CommRead, CommMode, CommState

4.7.2.10 Wählen

Syntax

```
Wählen(Nummer, Name: String)
Dial(Number, Name: String)
```

Erklärung

Die Funktion veranlasst die auf dem Rechner installierte Wähl-Anwendung die übergebene Nummer zu wählen. Der zweite Parameter ist optional. Windows verfügt über einen integrierten Telekommunikationsdienst, den TurboDB Studio über TAPI (Telephony Application Programming Interface) anspricht. Dieser Dienst stellt unter anderem ein Protokoll der durchgeführten Anrufe zur Verfügung. Im zweiten Parameter kann dazu der Name der anzurufenden Person mitgegeben werden. Das zuständige Modem bzw. die zuständige ISDN-Karte wird der Windows-Konfiguration entnommen (kann über Zubehör/Wahlhilfe konfiguriert werden), so dass keine weiteren Einstellungen vorzunehmen sind.

Beispiel

Das aktuelle Datenfenster sei ein Formular der Tabelle KUNDEN. Im Formular kann beispielsweise ein Makro-Schalter mit der Aufschrift Anrufen plaziert werden, der in seinen Eigenschaften unter *BeimAnklicken* den folgenden Aufruf ausführt:

```
Wählen(KUNDEN.Telefonnummer, KUNDEN.Name + ', ' + KUNDEN.Vorname)
```

4.7.3 Zwischenablage-Funktionen

4.7.3.1 Zwischenablage-Funktionen

Mit diesen Funktionen können Daten aus der Zwischenablage des Betriebssystems gelesen und auch hineingeschrieben werden.

AddStrToClipboard	Hängt eine Zeichenkette an den Inhalt der Zwischenablage
Clip2Text	Inhalt der Zwischenablage in Speicher-Datei kopieren
CopyStrToClipboard	Kopiert einen String in die Zwischenablage
Text2Clip	Inhalt der Speicher-Datei in Zwischenablage kopieren

4.7.3.2 AddStrToClipboard

Syntax

```
AddStrToClipboard(Zeichenkette: String)
```

Erklärung

Handelt es sich beim aktuellen Inhalt der Zwischenablage um einen Text, wird die übergebene Zeichenkette angehängt.

Beispiel

```
AddStrToClipboard(KUNDEN.Name + "," + KUNDEN.Vorname)
```

Siehe auch

CopyStrToClipoard, Zwischenablage-Funktionen

4.7.3.3 Clip2Text

Syntax

```
Clip2Text
```

Erklärung

Kopiert den Inhalt der Zwischenablage in die Speicher-Datei, wenn es sich um einen Text handelt.

Beispiel

Die Prozedur kopiert den Inhalt der Zwischenablage in das Memofeld Bemerkung des aktuellen Datensatzes.

```
Procedure Clipboard2Memo
      Clip2Text;
      ReadMemo(Bemerkung, 'RAMTEXT', 1);
EndProc
```

Siehe auch

Text2Clip, Zwischenablage-Funktionen

4.7.3.4 CopyStrToClipboard

Syntax

```
CopyStrToClipboard(Zeichenkette: String)
```

Erklärung

Kopiert die Zeichenkette in die Zwischenablage.

Beispiel

Die Adresse des aktuellen Kunden aus der Tabelle KUNDEN soll in die Zwischenablage kopiert werden:

```
CopyStrToClipboard(KUNDEN.Vorname + ' ' + KUNDEN.Name + Chr(13) + Chr(10) +
KUNDEN.Straße + Chr(13) + Chr(10) + KUNDEN.PLZ + ' ' + KUNDEN.Ort)
```

Chr(13)+Chr(10) ist der String für einen Zeilenvorschub. Auf diese Weise werden Name, Straße und Ort auf drei Zeilen verteilt.

Siehe auch

AddStrToClipboard, Zwischenablage-Funktionen

4.7.3.5 Text2Clip

Syntax

```
Text2Clip
```

Erklärung

Kopiert den Inhalt der Speicher-Datei in die Zwischenablage.

Beispiel

Die Prozedur kopiert den Inhalt des Memofeldes Bemerkung in die Zwischenablage.

```
procedure Memo2Clipbaord;
   CopyMemo(Bemerkung, 'RAMTEXT');
   Text2Clip;
endproc;
```

Siehe auch

Clip2Text, Zwischenablage-Funktionen

4.7.4 Dynamischer Datenaustausch

4.7.4.1 Dynamischer Datenaustausch

Mit dynamischem Datenaustausch (DDE) können Information zwischen Anwendungen ausgetauscht werden.

4.7.4.2 DDEExecute

Syntax

```
DDEExecute(Channel: Integer; Command: String)
DDEAusführen(Kanal: Integer; Befehl: String)
```

Erklärung

Eine Anweisung an eine andere Anwendung schicken. Zuvor muss ein DDE-Kanal mittels *DDEInitiate* mit der gewünschten Anwendung geöffnet werden. Die Anwendung muss die an sie gerichteten Anweisungen verstehen können, sonst tritt ein Fehler auf. Meistens handelt es sich bei den übergebenen Befehlen um Funktionen aus der Makrosprache des angesprochenen DDE-Servers.

Beispiel

siehe DDEInitiate

Siehe auch

DDEInitiate, DDETerminate, DDEPoke, DDERequest

4.7.4.3 DDEInitiate

Syntax

```
DDEInitiate(Application, Topic: String): Integer
DDEÖffnen(Anwendung, Thema: String): Integer
```

Erklärung

Initialisiert die Kommunikation mit einer anderen Anwendung über Dynamischen Datenaustausch (Dynamic Data Exchange). Dazu wird ein DDE-Kanal geöffnet. Bei Erfolg wird die Nummer des Kanals (>0) zurückgegeben. Alle weiteren DDE-Funktionen verwenden diese Kanalnummer. Schlägt das Öffnen eines Kanals fehl, wird 0 zurückgegeben.

Anwendung	Name der Anwendung mit der kommuniziert werden soll. Die Anwendung muss ein DDE-Server sein.
Thema	Ein dem Server bekanntes Thema. Für eine Textverarbeitung wäre dies zum Beispiel der Name eines Dokumentes, für eine Tabellenkalkulation der Name eines Rechenblattes

Das Thema System ist bei vielen DDE-Anwendungen zum Aufbau einer Verbindung immer bekannt. Anschließend können über *DdeRequest* weitere vorhandene Themen abgefragt werden.

Ein mit *DDEInitiate* geöffneter Kanal sollte unbedingt mittels *DDETerminate* geschlossen werden.

Beispiel

Die folgenden Zeilen starten die DDE-Kommunikation mit Word für Windows, laden ein Dokument und aktivieren die Druckvorschau:

```
vardef Channel: Integer;
Channel := DDEInitiate('WinWord', 'System');
if Channel > 0
      DDEExecute(Channel, '[DateiÖffnen .Name = "c:\dokument\peter.doc"]');
      DDEExecute(Channel, '[DateiSeitenansicht]');
      DDETerminate(Channel)
end
```

Siehe auch

DDETerminate, DDEExecute, DDEPoke, DDERequest

4.7.4.4 DDEPoke

Syntax

```
DDEPoke(Channel: Integer; Element, Data: String)
DDESchreiben(Kanal: Integer; Element, Daten: String)
```

Erklärung

Mit *DDEPoke* können Daten an ein Element einer anderen Anwendung geschickt werden. Dazu muss gewährleistet sein, dass die verbundene Anwendung das Element anhand des bei *DDEInitiate* übergebenen Themas identifizieren kann. Auf diese Weise kann beispielsweise die Zelle einer Tabellenkalkulation mit einem Wert belegt werden.

Beispiel

In die erste Zeile, erste Spalte des Rechenblattes Johann der Anwendung Excel wird der Wert *männlich* geschrieben:

```
VarDef Channel: Integer;
Channel := DDEInitiate('Excel', 'Johann.xls');
IF Channel > 0
      DDEPoke(Channel, 'Z1S1', 'männlich');
      DDETerminate(Channel)
END
```

Siehe auch

DDEInitiate, DDETerminate, DDEExecute, DDERequest

4.7.4.5 DDERequest

Syntax

```
DDEAbfragen(Kanal: Integer; Element: String):String
DDERequest(Channel: Integer; Element: String): String
```

Erklärung

Die Funktion fordert ein Element von der über den DDE-Kanal mit der Nummer Kanal verbundenen Anwendung an und gibt es als Zeichenkette zurück. Ein Element kann beispielsweise die Zelle eines Rechenblattes einer Tabellenkalkulation sein.

Das Element muss vom DDE-Server immer über das Thema zu identifizieren sein, das beim Öffnen des Kanals mittels *DDEInitiate* angegeben wurde.

Einige Anwendungen stellen standardmäßig das Thema System zur Verfügung. Wurde eine DDE-Verbindung für dieses Thema aufgebaut, können mit dem Element *SysItems* alle weiteren vorhandenen Themen abgefragt werden.

Beispiel

Das Beispiel ermittelt alle Elemente, die Word für Windows zum Thema System zur Verfügung stellt:

```
VarDef Channel: Real;
VarDef Elements: String;
Channel := DDEInitiate('WinWord', 'System');
IF Channel > 0
      Elements := DDERequest(Channel, 'SysItems')
      Message(Elements)
      DDETerminate(Channel)
END
```

Siehe auch

DDEInitiate, DDETerminate, DDEExecute, DDEPoke

4.7.4.6 DDETerminate

Syntax

```
DDETerminate(Channel: Integer)
DDESchließen(Kanal: Integer)
```

Erklärung

Schließen des DDE-Kanals mit der übergebenen Nummer. Einmal geöffnete DDE-Kanäle sollten

in jedem Fall geschlossen werden.

Beispiel

siehe DDEInitiate

Siehe auch

DDEInitiate, DDEExecute, DDEPoke, DDERequest

5 Datenbank Engine

dataweb bietet derzeit zwei Datenbank Engines an. TurboDB Native läuft auf allen 32-Bit Windows Plattformen und TurboDB Managed, für das .NET Framework und .NET Compact Framework.

TurboDB verfügt über einen sehr schnellen und kompakten 32 Bit Datenbank Kernel, der im Laufe der letzten acht Jahre fortlaufend an die Bedürfnisse der Anwendungs-Entwicklern angepasst wurde. Es sind keine Konfigurationsarbeiten zu erledigen, so dass sich eine Installation auf einfaches Kopieren der Dateien beschränkt. Dieses Merkmal macht TurboDB zur idealen Lösung für herunterladbare, CD-, DVD- oder Web-Anwendungen, die auf einem entfernten Webserver laufen sollen, der nur über FTP zu erreichen ist.

Dieses Buch beschreibt alle Features der TurboDB Datenbank Engine, die unabhängig von der verwendeten Entwicklungsumgebung und Komponenten-Bibliothek sind. Informationen zur VCL Komponenten-Bibliothek für Delphi und C++Builder finden Sie in "TurboDB 5 für VCL". Die Klassen des ADO.NET Providers für das .NET Framework sind in "TurboDB 5 für .NET" beschrieben.

Beide TurboDB Datenbank Engines zeichnen sich durch folgende Eigenschaften aus:

- Geringer Speicherbedarf
- Schnell
- Keine Installation
- Keine Konfiguration
- Mehrbenutzerfähigkeit
- Lizenz-freie Weitergabe erstellter Anwendungen
- Visueller Datenbank-Manager
- Viele weiter kostenfreie Werkzeuge

Die **Managed Engine** verfügt zusätzlich über folgende Vorteile:

- Läuft auf Windows für mobile Geräte: Pocket PC und Smartphone.
- Benötigt keine speziellen Rechte um in einer .NET Umgebung zu laufen (z.B. Unsafe Code)
- Unterstützt benutzerdefinierte Funktionen, Stored Procedures und benutzerdefinierte Aggregates.

Die **Native Engine** bietet diese Vorteile:

- Verschlüsselte Datenbank-Dateien
- Spezielle Spaltentypen für one-to-many und many-to-many Beziehungen zwischen Tabellen

5.1 Neuigkeiten und Upgrade

5.1.1 Neu in TurboDB Win32 v5

TurboDB 5 beinhaltet eine große Anzahl an neuen und erweiterten Features. Um einige dieser neuen Fähigkeiten nutzen zu können ist es nötig, bestehende Tabellen auf das neue Tabellen-Format 4 zu bringen. In den entsprechenden Sektionen der Dokumentation sehen Sie ob dies jeweils der Fall ist.

- Unkorrelierte und korrelierte Unterabfragen
- Das top Schlüsselwort begrenzt die Anzahl der Datensätze in der Ergebnis-Menge
- Starke Verschlüsselung entweder auf Blowfish oder Rijndael (AES) Algorithmus basierend
- Zeit-Felder mit einer Genauigkeit von Sekunden oder Millisekunden
- Vorgabewerte für Tabellen-Spalten
- Gültigkeitsbedingungen für Datensätze
- Das CASE Schlüsselwort
- Referentielle Integrität Primär- und Fremdschlüssel (auch kaskadierend)
- Ultra-schneller gewarteter Volltext-Index mit Ranking und verbesserter Integration in SQL
- Transaktions-Unterstützung mit automatischem Crash Recovery
- Verbesserte typisierte aggregierte Spalten
- Join-Bedingungen können jetzt beliebige Such-Bedingungen sein, mit Berechnungen und logischen Operatoren, nicht nur Vergleiche
- UNION, EXCEPT und INTERSECT
- Nullable für Zeichenketten-Typen
- Verwaltete Datenbanken mit Datenbank-weiten Eigenschaften
- Die CAST Funktion
- Optimierung der Seiten-Größen gemäß der eingestellten Kapazität (Tabellen-Level 4)
- Bis zu 50 Indexe pro Tabelle (Tabellen-Level 4)
- Benennung der Index-Dateien nach dem Schema <Tabellenname>|<Indexname>.ind um Namenskonflikte zu vermeiden (Tabellen-Level 4)
- Tabellen und Spalten Namen können jetzt Leerzeichen und andere Sonderzeichen enthalten (Tabellen-Level 4)

5.1.2 Upgrade auf TurboDB Win32 v5

Bei der Umstellung von TurboDB 4 auf TurboDB 5 sind einige Punkte zu beachten:

Wichtiger Hinweis

Beachten Sie bitte, dass TurboDB 5 und TurboDB 4 nicht gleichzeitig auf Datenbank Tabellen zugreifen können. Die Meldung "Error in log-in" wäre die Folge. Falls diese Meldung angezeigt wird obwohl keine TurboDB 4 Anwendung läuft, sind noch net/rnt/mov/rmv Dateien vorhanden. Löschen Sie diese Dateien und alles wird wie erwartet funktionieren.

Neu reservierte Schlüsselwörter

Die folgenden Bezeichner sind neue reservierte Schlüsselwörter in TurboDB 5. Falls Ihr Datenbank-Schema eines dieser Wörter als Tabellen- oder Spalten-Name verwendet, müssen sie diese in SQL-Statements entweder mit doppelten Anführungszeichen umschließen oder umbenennen:

ACTION, ALL, ANY, CASCADE, CASE, CAST, DICTIONARY, ENCRYPTION, EXCEPT, EXISTS, FULLTEXTINDEX, INTERSECT, NO, SOME, TOP, UNION, WHEN

Verwendung von Anführungszeichen

TurboDB 5 verwendet einfache Anführungszeichen ausschließlich für Zeichenketten Literale. In TurboDB 4 waren doppelte Anführungszeichen hierfür ebenso erlaubt und müssen nun ausgetauscht werden. Doppelte Anführungszeichen sind nun ausschließlich für Bezeichner zu verwenden und ermöglichen es mit Namen zu arbeiten, die Leerzeichen oder Sonderzeichen beinhalten oder mit reservierten Schlüssel-Wörtern übereinstimmen.

Verschlüsselung

TurboDB bietet nun zusätzliche Methoden für eine sicher Verschlüsselung. Aus diesem Grund hat sich die Syntax für die Statements create table und alter table geändert.

Volltext Indexe

Die Volltext Indizierung wurde grundlegenden neu gestaltet und ist nun wesentlich schneller und

dazu gewartet. Die alten Volltext Indexe werden für die alten Tabellen-Formate natürlich weiter unterstützt. Wenn Sie aber Ihre Tabellen upgraden, müssen Sie den Volltext-Index-Teil Ihrer SQL-Statements anpassen.

5.1.3 Neu in TurboDB Managed v2

Version 2.0

- Das neue Prädikat CONTAINS in Verbindung mit der neuen Volltext-Indizierung ermöglicht eine mächtige und sehr schnelle Suche nach beliebigen Wörtern.
- Tabellen-Namen können bis zu 79 Zeichen lang, Spaltennamen bis zu 128 Zeichen lang sein.
- Einführung eines Connection-Pools zur Verbesserung der Performanz.
- Unterstützung von Fast Encryption und Rijndael (AES).
- Mit der Klasse *TurboDBConnection* wird die Komprimierung der Datenbank möglich
- UNION, INTERSECT und EXCEPT Statements.
- TurboDB Pilot mit stark verbessertem Komfort bei der Verwaltung von Datenbanken und Kommandos.
- TurboDB Pilot mit visuellem Designer zum Erstellen und Ändern von Datenbank-Tabellen.

Version 1.3

TurboDB Managed 1.3 beinhaltet viele neue Features rund um die Programmierung.

- Benutzerdefinierte Funktionen ermöglichen die Implementierung von eigenen SQL Funktionen entweder in SQL oder als .NET Assembly.
- Mit Stored Procedures können komplexe Befehlssequenzen mit einem einzigen Aufruf abgearbeitet werden. Sie werden entweder in TurboSQL oder in einer beliebigen .NET Sprache implementiert.
- User-defined Aggregates erlauben die Definition neuer Aggregatsfunktionen in einer beliebigen -NET Sprache zur Verwendung in SQL Statements.

Lesen Sie das Kapitel "Programmiersprache" für weitere Informationen.

5.1.4 Upgrade auf TurboDB Managed v2

TurboDB Managed 2.0 ist voll-kompatibel zu Version 1.x.

Neue reservierte Schlüsselwörter

Es wurden einige neue Schlüsselwörter eingeführt. Wurde eines davon als Tabellen- oder Spalten-Name verwendet, muss dieser in Statements nun in eckigen Klammern [...] oder Anführungszeichen "..." geschrieben werden: BY, CORRESPONDING, ENCRYPTION, EXCEPT, INTERSECT, UNION, CONTAINS, FULLTEXTINDEX.

5.2 TurboDB Engine Konzepte

Dieses Kapitel beschreibt die grundlegenden Konzepte der TurboDB Database Engines.

Überblick behandelt Leistungsmerkmale, Namenskonventionen und Datentypen.

Datenbanken erklärt den Unterschied zwischen Single-File und Directory Datenbanken.

Indexe behandelt die verschiedenen von TurboDB unterstützten Index-Arten.

Automatic Data Linking präsentiert das TurboDB Konzept für ein schnelleres und weniger Fehler-anfälliges Datenmodell.

Mehrbenutzerzugriff und Sperren beschreibt wie TurboDB Multi-User Access implementiert.

Optimierung bietet eine Liste von Punkten, die Sie prüfen können falls Ihre Anwendung

nicht schnell genug ist.

Fehlerbehandlung beschreibt welche Fehler bei der Arbeit mit der Datenbank auftreten können und wie diese zu behandeln sind.

Verschiedenes behandelt die physikalische Datenbanktabellen, Datensicherheit und Lokalisierung.

5.2.1 Überblick

Leistungsmerkmale
Tabellen- und Spaltennamen
Datentypen für Tabellenspalten

5.2.1.1 Kompatibilität

Es gibt zwei Implementierungen der TurboDB Datenbank Engine, die Win32 Engine und die .NET Engine, letztere auch als Managed Engine bezeichnet. Die beiden Datenbank Engines können mit identischen Datenbank-Dateien arbeiten, solange die Einschränkungen der beiden Engines berücksichtigt werden. Das sind die zu beachtenden Regeln, wenn Sie Datenbank-Dateien sowohl mit TurboDB Managed 2.x als auch mit TurboDB für Win32 5.x bearbeiten wollen.

- Verwenden Sie eine Single-File- statt einer Directory-Datenbank
- Verwenden Sie maximal 40 Zeichen für Tabellen- und Spaltennamen.
- Verwenden Sie nicht *Blowfish* Verschlüsselung, sondern *FastEncrypt* oder *Rijndael*. Das Passwort für alle Tabellen muss identisch sein, da TurboDB Managed nur ein Passwort für eine Datenbank unterstützt.
- TurboDB für Win32 ignoriert benutzerdefinierte Funktionen und Stored Procedures. Sie können weder in berechneten Indexen noch in Queries benutzt werden.
- Verwenden Sie keine Sprachtreiber.
- Da sich der Sperrmechanismus unterscheidet können TurboDB Managed und TurboDB für Win32 nicht dieselbe Datenbank gleichzeitig öffnen. Ein abwechselnder Zugriff ist möglich.

5.2.1.2 Leistungsmerkmale

Einige technische Daten zu Level 4 und 5 Tabellen:

Maximale Anzahl Zeilen pro Tabelle	2 G
Maximale Tabellengröße	4 EB (ein Exabyte ist 1G mal 1GB)
Maximale Anzahl Spalten pro Tabelle	1.000
Maximale Größe einer Zeile	32 KB
Maximale Anzahl Benutzer-definierter Indexe pro Tabelle	48 (ab Tabellen-Level 4) oder 13 (bis Tabellen-Level 3)
Maximale Anzahl Ebenen in hierarchischen Indexen	255
Maximale Größe der Index-Information	32 KB
Maximale Anzahl Tabellen pro Datenbank	255
Maximale Länge einer String-Spalte	32.767
Maximale Anzahl von Link-Feldern pro Tabelle	9
Maximale Gesamtgröße aller Blobs einer Tabelle	1 TB (bei Blockgröße 512 B) bis 64 TB (bei Blockgröße 32 KB)

5.2.1.3 Tabellen- und Spaltenamen

Bezeichner

Bezeichner bestehen aus einem echten Buchstaben gefolgt von alphanummerischen Zeichen, dem Unterstrich_ und dem Bindestrich. Sie dürfen maximal 40 Zeichen lang sein. Deutsche Umlaute sind erlaubt.

Gültige Bezeichner sind:

```
Name
Alter
Straße
Durchschitts_Dauer
Tag_der_Geburt
Adresse8
Lösung
```

Spalten- und Tabellen-Namen

Spalten- und Tabellen-Namen können alle ANSI-Zeichen beinhalten außer Steuercodes, eckige Klammern [] und doppelte Anführungszeichen. Sie sind auch maximal 40 Zeichen lang und müssen mit einem echten Buchstaben beginnen. Entspricht der Name den Regeln für Bezeichner, kann er in allen ausdrücken und Statements ganz normal benutzt werden. Falls nicht, muss er entweder in doppelten Anführungszeichen oder in eckige Klammern gesetzt werden.

Diese Beispiele sind **ungültige** Bezeichner, die trotzdem als Spalten-Namen verwendet werden können, indem sie in doppelte Anführungszeichen oder eckige Klammern gesetzt werden:

```
Anzahl Einträge(Leerzeichen nicht erlaubt)
3645 (das erste Zeichen muss ein Buchstabe sein)
```

Die maximale Länge für Tabellen- und Spaltennamen ist 40 für Tabellen mit Level <= 4. Ab Tabellenlevel 5 können Tabellennamen bis zu 79 Zeichen und Spaltennamen bis zu 128 Zeichen lang sein.

5.2.1.4 Datentypen für Tabellenspalten

TurboDB bietet die folgenden Datentypen für Tabellen-Spalten:

String

Ein String Feld beinhaltet alphanummerische Zeichen bis zu einer bestimmten Länge. Die maximale Länge ist 32.767 Zeichen (255 Zeichen bis Tabellen-Level 3). Ein String Feld stellt ein Byte pro Zeichen zur Verfügung plus ein oder zwei Byte für die Länge der Zeichenkette, plus ein Byte falls die Spalte über einen Null-Indikator verfügt.

WideString

Bis zu 32767 Unicode Zeichen (255 bis Tabellen-Level 3). Da ein Unicode Zeichen 2 Bytes benötigt, ist die resultierende Feldgröße zwei mal die Anzahl der Zeichen, plus 2 Byte für die Länge, plus ein Byte falls das Feld über einen Null-Indikator verfügt.

Byte

Zahlen von 0 bis 255. Byte Felder können optional einen Null Indikator haben. Die Größe ist ein oder zwei Bytes, abhängig davon ob ein Null Indikator verwendet wird oder nicht.

SmallInt

Zahlen von -32767 bis +32768. SmallInt Felder können einen optionalen Null-Indikator haben. Die Größe ist zwei oder drei Bytes.

Integer

Zahlen von -2147483648 bis +2147483647. Integer Felder können einen optionalen Null Indikator haben. Es werden vier oder fünf Bytes in der Datenbanktabelle benötigt.

LargeInt

Zahlen von –2^63 bis +2^63 – 1 mit einem optionalen Null Indikator. Ein Int64 Feld verwendet acht oder neun Byte in der Tabelle.

Float

Speichert eine 8 Byte Fließkommazahl, z.B. von 5.0e-324 bis 1.7 x 10e308. Ein optionaler Null Indikator ist möglich. Die Größe ist acht oder neun Bytes.

Zeit

Für Tabellen-Level 4: Werte von 00:00:00.000 bis 23:59:59.999. Die Genauigkeit wird bei der Erstellung der Spalte festgelegt und beträgt entweder Minuten, Sekunden oder Millisekunden. Für Tabellen-Level bis 3 ist die Genauigkeit immer Minuten. Ein optionaler Null-Indikator ist möglich. Die Größe ist, abhängig von der Genauigkeit und dem Null-Indikator, zwei bis fünf Bytes.

Datum

Werte von 1.1.0000 bis 31.12.9999. Das Feld benötigt vier Bytes. Intern werden Datumsfelder als gepacktes Bitfeld dargestellt.

DateTime

Werte von 1.1.0000 00:00 bis to 31.12.9999 23:59. Es werden acht Byte benötigt.

Boolean

Speichert die logischen Werte True oder False. Ein optionaler Null Indikator ist möglich. Die Größe ist ein oder zwei Bytes.

Auswahl

Ein Auswahl Typ definiert eine Liste von lesbaren, aussagekräftigen Ausdrücken und speichert den Wert in einem einzigen Byte in der Tabelle ab. Es ist dasselbe Prinzip wie beim Aufzählungstyp von Delphi, läßt sich aber direkt auf Datenbank Tabellen anwenden.
Sie können zum Beispiel ein Feld mit Namen Geschlecht haben und die Aufzählungswerte könnten mit weiblich, männlich und unbekannt definiert sein. Auf diese Weise haben Sie automatisch einen verständlichen und sicheren Weg die Werte des Feldes anzuzeigen und zu editieren und gleichzeitig werden die Werte sehr effektiv abgespeichert. Aufzählungswerte können selbstverständlich auch in Filter- und Suchausdrücken verwendet werden: 'Geschlecht = männlich'.
Ein Aufzählungswert darf bis zu 20 Zeichen lang sein, bis zu 15 Aufzählungswerte sind möglich und die Gesamtlänge aller Aufzählungswerte zusammen inklusive Trennzeichen dazwischen darf nicht länger als 255 Zeichen sein.

Memo

Lange Zeichenketten mit variabler Länge (bis 1G). Memos werden in einer zusätzlichen Datei gespeichert, die denselben Namen hat wie die Datenbanktabelle, aber mit der Erweiterung mmo.

Wid**e**Memo

Unicode String variabler Länge (bis 1G). Wide Memos werden in BLOBs gespeichert (Dateierweiterung blb).

Blob

Bilder und andere binäre Daten variabler Länge. Blobs werden in einer zusätzlichen Datei gespeichert, die denselben Namen hat wie die Datenbanktabelle, aber mit der Erweiterung blb.

Link

Link-Felder sind Zeiger auf Datensätze in einer anderen Tabelle. (1:n Beziehung). Linkfelder werden in Die "Automatic Data Link Technologie" beschrieben. Ein Linkfeld speichert den Wert des AutoInc-Feldes des Datesatzes auf den das Link-Feld verweist. Seine Größe ist vier Bytes.

Relation

Relationsfelder enthalten virtuell eine Liste von Zeigern, auf diese Weise ermöglichen sie n:m Beziehungen. Relationen werden in Die "Automatic Data Link Technologie" beschrieben. Physikalisch werden Relationfelder durch eine zusätzliche Relationstabelle realisiert, die über zwei Linkfelder verfügt um die n:m Beziehung abzubilden. Daher benötigt ein Relationsfeld selbst 0 Bytes.
Nicht verfügbar in TurboDB Managed

AutoInc

Automatisch generierte, eindeutige 32-Bit Zahl. AutoInc Felder werden von der Datenbank Engine verwaltet und können daher nicht editiert werden. Die "Automatic Data Link Technologie" verwendet AutoInc Felder als Primärindex um Referenzen auf Datensätze zu speichern.

GUID

128 Bit Zahl, die von MS COM und ActiveX Technologien verwendet wird. GUID steht für Globally Unique Identifier. GUIDs werden für gewöhnlich durch eine Betriebssystem-Funktion berechnet, die sicherstellt, dass dieser Wert weltweit einzigartig ist.

5.2.2 Datenbanken

TurboDB bietet zwei verschiedene Arten von Datenbanken.

Single-File vs. Datenbank-Verzeichnisse

Datenbank-Verzeichnisse

Datenbank-Verzeichnisse sind Verzeichnisse auf der Festplatte, die alle unterschiedlichen TurboDB Datenbank-Objekte in einzelnen Dateien enthalten. Datenbank-Verzeichnisse werden von TurboDB schon immer unterstützt.

Singe-File Datenbank

Eine Singe-File Datenbank ist eine einzelne Datei, die alle Objekte der Datenbank beinhaltet. Ein Datenbank-File hat die Datei-Erweiterung *.tdbd. Single-File Datenbanken werden von TurboDB seit Version 4 unterstützt. Sie haben den Vorteil, der sehr einfachen Weitergabe und können unproblematisch auf der Festplatte kopiert und verschoben werden.

Der Vorteil von Datenbank-Verzeichnissen dagegen ist, dass der Zugriff ein bisschen schneller ist einzelne Tabellen in mehreren Datenbanken verwendet werden können.

Single-File Datenbanken werden über ein virtuelles Dateisystem realisiert, das von dataWeb implementiert wurde. Diese Schicht bildet die Datenbank-Objekte entweder auf die verschiedenen Dateien in einem Verzeichnis oder auf eine einzelne Datenbank-Datei ab. dataWeb bietet dazu ein Werkzeug an, den dataWeb Compound File Explorer, das Datenbank-Dateien öffnen, den Inhalt anzeigen kann und editieren. Das ist ein Weg um Verzeichnis-basierte Datenbanken in eine Single-File Datenbank zu überführen.

Während TurboDB Native beide Datenbank-Typen unterstützt, kann TurboDB Managed ausschließlich mit Single-File Datenbank arbeiten.

Datenbankkataloge

In früheren Versionen waren TurboDB Datenbanken lediglich Sammlungen von Datenbanktabellen, die in separaten Dateien abgespeichert wurden. Während dieser Ansatz den Vorteil hat, dass einzelne Tabellen Bestandteil mehrerer Datenbanken sein können, gibt es auch einige Nachteile. Einer ist, dass der Name einer Tabelle nicht immer aufgelöst werden kann, da sich die Tabellendatei in einem weit entfernten Verzeichnis befinden kann. Falls die einzelnen Tabellen auf unterschiedliche Weise verschlüsselt sind ist ein weiterer Nachteil, dass verschiedene Passwörter benötigt werden um die Datenbank zu öffnen

Daher führt TurboDB 5 für Win32 Datenbankkataloge ein, die eine Liste der Tabellen und zusätzliche Datenbank-weite Eigenschaften speichern. Kataloge sind sehr hilfreich bei Verwendung von Verzeichnis-Datenbanken. Für Single-File Datenbanken existieren die genannten Nachteile nicht, oder sind weniger problematisch.

TurboDB Managed unterstützt ausschließlich Single-File Datenbanken, verfügt über alle nötigen Informationen und bietet daher keine explizite Katalogfunktionalität.

Hinweis: Datenbanken mit Katalogfunktionalität wurden in früheren Versionen als verwaltete (managed) Datenbanken bezeichnet. Dieser Terminus führt aber zu einem Namenskonflikt mit dem Produkt TurboDB Managed.

5.2.2.1 Sessions und Threads

Seit TurboDB Version 4 muss eine Session erzeugt werden, bevor Tabellen und Queries geöffnet werden können. Falls Sie eine Komponenten-Bibliothek verwenden (TurboDB für VCL/CLX oder .NET) werden Sessions im Datenbank- oder Connection-Objekt versteckt verwaltet.

Sie können so vielen Sessions erzeugen wie Sie möchten, sollten aber die Konsequenzen einer Multi-Session Anwendung kennen:

- Cursor der gleichen Tabelle in verschiedenen Sessions werden auf Dateiebene synchronisiert. Das ist wesentlich langsamer als die Synchronisation der Cursor in einer Session, die im Speicher durchgeführt wird.
- Sie können verschiedene Threads für verschiedene Sessions einsetzen, aber nicht verschiedene Threads für eine Session. Aus Gründen der Performanz gibt es keine eingebaute Thread-Synchronisation innerhalb einer Session.

5.2.2.2 Tabellen-Levels

Im Zuge des Ausbaus und der Verbesserung von TurboDB wurden im Laufe der Zeit verschiedene Speicherformate entwickelt. Diese Format werden Tabellen-Level genannt und im folgenden beschrieben. Während TurboDB Native alle Level unterstützt, kann TurboDB Managed nur mit Level 5 und höher arbeiten.
TurboDB Studio 4 arbeitet derzeit nur mit Tabellen bis Level 3.

Table Level 5

- Ist kompatibel mit TurboDB Managed.
- Unterstützt Gültigkeitsbedingungen, Vorgabewerte, berechnete Spalten und Indexe in Standard SQL.
- Unterstützt gleichnamige Tabellen in unterschiedlichen Datenbanken im selben Verzeichnis.
- Unterstützt die Verschlüsselung des Tabellenschemas.
- Ist vorbereitet für die Unterstützung von Zeichensätzen.
- Ist vorbereitet für Unicode Tabellen- und Spaltennamen.

Table Level 4

- Unterstützt Primärschlüssel und eindeutige Schlüssel.
- Unterstützt Zeit-Spalten mit einer Genauigkeit bis zu Millisekunden.
- Unterstützt zusätzliche Verschlüsselungs-Algorithmen (strong encryption).
- Unterstützt Gültigkeitsbedingungen und Fremdschlüssel.
- Passt die Sortierung an den SQL Standard an, so dass Null-Werte kleiner als alle anderen sind.
- Erhöht die Zahl der Indexe, die für eine Tabelle möglich sind.
- Unterstützt sichere Verschlüsselung.
- Ermöglicht gewartete Volltext-Indexe.

Table Level 3

- Neu sind Unicode Strings, DateTime and GUID Spalten.

Table Level 2

- Führt 32 Bit Integer und ANSI Strings ein.

Table Level 1

- Das erste Datei-Format. Kompatibel mit TurboDB für DOS.

5.2.3 Indexe

Indexe werden zusätzlich zu den eigentlichen Tabellen verwaltet, um schnelles Suchen und Sotieren zu ermöglichen. TurboDB Indexdefinitionen bestehen entweder aus einer Liste von Feldnamen oder einem Ausdruck. Falls ein Index als eindeutig definiert ist, werden Datensätze die diese Eindeutigkeit verletzen nicht akzeptiert. Eine weitere Form eines Index ist der Volltextindex.

Indexe basierend auf einer Liste von Feldern

Diese Indexe werden in der Reihenfolge des ersten Feldes in der Feldliste sortiert. Wenn zwei Datensätze den gleichen Wert für das erste Feld haben, werden sie nach dem zweiten Feld der Feldliste sortiert und so weiter. Es kann bis 10 Felder in der Indexfeldliste geben. Jedes Feld kann in auf- oder absteigender Reihenfolge in die Indexbeschreibung eingehen.

Indexe basierend auf einem Ausdruck

Diese Indexe werden nach dem Wert eines beliebigen Ausdrucks sortiert, der bis 40 Zeichen lang sein kann. Ist der Ausdruck vom Type String, wird der Index wie die Werte einer alphanummerischen Spalte sortiert. Wenn der Ausdruck nummerischer Art ist, wird der Index entsprechend der normalem nummerischen Reihenfolge sortiert.

Volltext-Indexe

Ein Volltext-Index erlaubt es einem Anwender in jedem Feld der Tabelle nach einem Schlüsselwort oder einer Reihe von Schlüsselwörtern zu suchen. Volltext-Index benötigen eine separate Tabelle,

die Wörterbuch-Tabelle, welche die indizierten Wörter enthält. Volltext-Indexe sind für Tabellen-Level 4 anders implementiert als für die niedrigeren Level. In Level 4 gibt es nur ein Speicherobjekt, das die Verbindung zwischen Wörterbuch-Tabelle und indizierter Tabelle herstellt. Die Dateierweiterung dieses Objekt ist fti. In den älteren Tabellen-Formaten wurde die Verbindung über Relations-Felder hergestellt, was eine zusätzliche Datenbank-Tabelle (Extension rel) und zwei weitere Indexe (in1 und in2) nötig macht.

Indexe können mit verschiedenen TurboDB Tools zur Entwurfszeit erzeugt, geändert und gelöscht werden. Zur Laufzeit kann TurboSQL oder Methoden der VCL Komponenten Bibliothek verwendet werden um Volltext-Index zu erstellen, zu erneuern oder zu löschen.

5.2.4 Automatic Data Link

In den allermeisten Fällen sind Tabellenverknüpfungen für alle Abfragen gleich. Z.B. gehören einzelne Posten immer zu einer Rechnung, Autoren sind immer mit den Büchern verknüpft, die sie geschrieben haben u.s.w. Daher ermöglicht es TurboDB die verschiedenen Arten von Verknüpfungen von einer Tabelle zu anderen Tabellen in der Tabelle selbst zu definieren.

Stellen Sie sich vor, Sie haben eine Rechnungstabelle, die das Rechnungsdatum, die Kundennummer, die Rechnungsnummer und andere rechnungsbezogene Informationen enthält. Die einzelnen Rechnugsposten werden in einer anderen Tabelle eingetragen, die Felder für die Artikelnummer, den Preis, den Gesamtbetrag und andere enthält. Wie verknüpfen Sie den Einzelposten mit der Rechnung deren Bestandteil der Posten ist? Der traditionelle Weg ist es in der Postentabelle ein zusätzliches Feld zu haben, das die Nummer der Rechnung aufnimmt, zu der der Posten gehört. Jede Abfrage, die die Rechnungs-Posten Beziehung berücksichtigt, muss die folgende Bedingung formulieren: ...where "POSTEN.RechnungsNummer" = "RECHNUG.RechnungsNummer"...

Was ist das?

Sie können diesen traditionellen Weg auch mit TurboDB gehen, die bevorzugte Lösung ist ein bisschen anders. Anstatt ein RechnungsNummer Feld in der POSTEN Tabelle zu haben würden Sie lieber einen Zeiger auf die Rechnungstabelle verwenden. Dieser Zeiger wird als Linkfeld bezeichnet. Da TurboDB standardmäßig eine (eindeutige) RecordId in jede neue Tabelle einfügt, speichert das Linkfeld der Postentabelle nur die RecordId der Rechnung zu der der Posten gehört. Da die Definition des Linkfeldes die Information beinhaltet, dass der Inhalt dieser Spalte auf einen Eintrag in der Rechnungstabelle zeigt, weiß die Datenbank über diese Beziehung bescheid und wird dies standardmäßig bei jeder Abfrage berücksichtigen.

Diese Art Tabellen zu verknüpfen hat einige große Vorteile:

Abfragen über verknüpfte Tabellen sind sehr einfach, da das System "weiß" wie die Tabellen miteinander zu verknüpfen sind. Die Abfragen können viel schneller sein, weil eine RecordId nur eine Zahl ist, während Sekundärindexe oftmals wesentlich komplexer sind. Das Ändern von Indexen, Spaltennamen oder Spaltentypen beläßt die Verknüpfung unberührt. Über eine spezielle Link Notation ist es sehr einfach auf den Masterdatensatz zuzugreifen.

Man kann diese Strategie als den objektorientierten Weg betrachten mit Datenbanktabellen zu arbeiten. Sie ist nicht streng konform mit dem relationalem Paradigma, bringt aber das Feeling von Zeigern und Referenzen ins Spiel. Der Rechnungsposten "weiß" zu welcher Rechnung er gehört. Diese Verbindung liegt in der Natur der Sache und wird sich sicher nicht sehr oft ändern.

Ein weiterer Vorteil ist, dass Link- und Relationsfelder dem Anwender nicht nur die rein technischen AutoInc-Werte anzeigen können. Jeder AutoInc-Spalte kann eine Anzeigeinformation bestehend aus Werten anderer Spalten zugeordnet werden. Diese wird von Link und Relationsfeldern anstelle der nummerischen Werte angezeigt. Hier ein Beispiel:

CREATE TABLE DEPARTMENT (Name CHAR(20), Id AUTOINC(Name)

CREATE TABLE EMPLOYEE (LastName CHAR(40), Department LINK(DEPARTMENT))

Die Abfrage

SELECT * FROM EMPLOYEE

liefert eine Liste von Nachnamen und Abteilung-Namen, das der Name der Abteilung als Anzeigeinformation für die AutoInc-Spalte *Id* definiert ist.

Wie funktioniert es?

Da Linkfelder auf so einfache Weise 1:n Beziehungen (eine Rechnung hat mehrere Posten)

einführen, wirft dieses objektorientierte Konzept die Frage nach m:n Beziehungen als eine Liste von Zeigern auf, die von einer Tabelle auf eine andere zeigen. TurboDB Relationsfelder sind die Antwort auf diese Frage. Eine Tabelle, die ein Relaionsfeld enthält verknüpft einen Datensatz mit mehreren Datensätzen der anderen Tabelle und umgekehrt. Um zu dem Beispiel mit Autoren und Büchern zurückzukehren, würde das Einfügen eines Relationsfeldes in die Büchertabelle die Tatsache berücksichtigen, dass ein Buch von mehreren Autoren geschrieben und ein Autor durchaus an mehreren Büchern beteiligt sein kann.

Wie Sie sicher schon annehmen sind Relationsfelder nicht so einfach zu implementieren wie Linkfelder. M:n Beziehungen werden durch eine zusätzliche Zwischen-Tabelle realisiert, die einen Datensatz für jede Verknüpfung zwischen den beiden Tabellen erhält. Das genau ist es was TurboDB macht, wenn Sie ein Relationsfeld in die Buchtabelle einbauen das auf die Autorentabelle zeigt. TurboDB wird eine versteckte Zwischen-Tabelle erzeugen, mit einem Linkfeld auf die Autorentabelle und einem Linkfeld auf die Büchertabelle. Das ist es auch was Sie tun müssten wenn Sie auf traditionelle Weise arbeiten würden. Aber mit TurboDB wird die Zwischen-Tabelle automatisch und transparent erzeugt und gepflegt.

Kompatibilität

Dieses Feature wird nur zum Teil in TurboDB Managed unterstützt. In TurboDB Managed gibt es momentan Link-Spalten, aber keine Relations-Spalten.

5.2.4.1 Mit Link- und Relationsfelder arbeiten

Um von der Automatic Link Technologie zu profitieren, sollten Sie darüber nachdenken in jeder Tabelle, die Sie neu anlegen, mit Link- oder Relationsfeldern zu arbeiten. Sie werden es sehr bald als völlig natürlich empfinden die Verknüpfungsinformation in der Tabelle zu speichern. Schließlich machen Sie ja dasselbe mit Ihren Delphi, C++ oder Java Klassen, oder?

Hnizufügen von Link- und Relations-Spalten

Wenn Sie mit Link- oder Relations-Felder arbeiten möchten, um eine 1-n oder n-m Beziehung zwischen Tabellen herzustellen, müssen Sie entscheiden welche der Tabellen die Quelle und welcher das Ziel der Beziehung ist. Erstere wird als Kind-Tabelle, zweitere als Eltern-Tabelle bezeichnet. Es verhält sich wie mit der referenzierenden und referenzierten Tabelle, wenn Sie an die Arbeit mit traditionellen Fremdschlüsseln denken.

Die Eltern-Tabelle muss über eine AutoInc-Spalte verfügen,. die als Primärschlüssel für die Verknüpfung dient. Die Kind-Tabelle muss eine Link- oder Relations-Spalte beinhalten, welche die Verknüpfung etabliert. Eine Link-Spalte kann genau einen Zeiger auf die Eltern-Tabelle speichern. Der Zeiger wird als AutoInc-Wert der Eltern-Tabelle angezeigt oder als den Werten der Tabellenspalten der Anzeigeinformation, falls diese für die AutoInc-Spalte der Eltern-Tabelle definiert ist. Relationsfelder können mehrere Zeiger auf die Eltern-Tabelle speichern, die als Liste der AutoInc-Werte oder als Liste der Anzeigeinformationen angezeigt werden.

Link- und Relationsfelder beim direkten Tabellen-Zugriff

(Direkter Tabellenzugriff ist mit der VCL/CLX-Komponenten-Bibliothek möglich, nicht aber mit ADO.NET.)

Wenn Sie Ihre Links und Relationen erst einmal definiert haben, werden diese bei jeder Abfrage von der Datenbank berücksichtigt. Sogar wenn Sie überhaupt kein Suchkriterium angeben, werden wirklich nur verknüpfte Detail-Datensätze zum aktuellen Master-Datensatz angezeigt. Für den seltenen Fall, dass Sie dieses Verhalten nicht möchten, können Sie jederzeit einen anderen Equate Join angeben, der den Standard überschreibt.

Link- und Relationsfelder in TurboSQL

In TurboSQL-Abfragen werden die durch Link- und Relationsfelder definierten Verknüpfungen nicht automatisch hergestellt. Mit einem einfachen JOIN erhält man allerdings den Bezug:

```
SELECT * FROM Master JOIN Detail ON Detail.LinkField = Master.RecordId
```

Zum Eintragen neuer Datensätze in den Verbund, kann man die Funktion CurrentRecordId verwenden:

```
INSERT INTO Master VALUES(...); INSERT INTO Detail VALUES(...,
CurrentRecordId(Master), ...)
```

Mit diesem zusammengesetzten Statement wird zuerst ein Datensatz in die Master-Tabelle eingetragen und dann sofort ein weiterer Datensatz in die Detail-Tabelle, wobei als Wert für das Link-Feld die letzte RecordId der Master-Tabelle benutzt wird. Dadurch ist der Detail-Datensatz mit

dem Master-Datensatz verknüpft.

Komaptibilität

Dieses Feature wird nur zum Teil in TurboDB Managed unterstützt. In TurboDB Managed gibt es momentan Link-Spalten, aber keine Relations-Spalten.

5.2.5 Mehrbenutzerzugiff und Sperren

Standardmäßig öffnet TurboDB Tabellen im Shared Mode. Es können also mehreren Anwendungen gleichzeitig auf eine Datenbanktabelle zugreifen. Um inkonsistente Änderungen der Tabelle zu vermeiden, gibt es einen transparenten Sperrmechanismus, der Datensätze für die Dauer des Editiervorganges durch eine Anwendung sperrt.

Einige Operationen benötigen allerdings exklusiven Zugriff auf eine Datenbanktabelle und werden daher zurückgewiesen falls eine weitere Anwendung die Tabelle benutzt. Diese Operationen sind:

- Ändern der Tabellenstruktur (AlterTable)
- Löschen einer Tabelle (DeleteTable)
- Löschen eines Index

Lock Files

Da TurboDB eine dateibasierte Datenbank Engine ist, werden Tabellensperren über eigene Verwaltungsdateien verwaltet. Diese Lock Dateien haben denselben Namen wie die entsprechende Datenbanktabelle, aber mit der Dateierweiterung "net". Die Lock Datei beinhaltet eine Liste aller Anwendungen, die auf die Datenbanktabelle zugreifen und verwaltet die unterschiedlichen Sperren auf diese Tabelle.

Wenn eine Anwendung in eine Datenbanktabelle schreiben will, muss sie erst in der Lock Datei nachsehen, ob es erlaubt ist. Falls ja, registriert sich die Anwendung in der Lock Datei als schreibend, führt die Aktion aus und meldet sich wieder ab. Die Lock Datei wird von der ersten Anwendung erzeugt, die auf die Tabelle zugreift und wieder gelöscht, wenn die letzte Anwendung den Zugriff beendet. Daher verfügt eine Tabelle über keine Lock Datei, solange sie nicht geöffnet wird.

Bemerkung: Falls eine Anwendung abstürzt oder beendet wird, während eine Sperre auf die Tabelle oder einen Datensatz eingerichtet ist, kann die Lock Datei nicht gelöscht werden und die Sperre bleibt bestehen. Das kann gerade beim Debuggen einer TurboDB Anwendung des öfteren passieren, da ein Reset während eine Sperre eingerichtet ist genau diesen Effekt hat. Seit Version 4 hat TurboDB einen einzigartigen Mechanismus, der solche toten Sperren erkennt und automatisch aufhebt.

5.2.5.1 Tabellensperren

TurboDB Engine arbeitet mit zwei verschiedenen Arten von Tabellensperren:

Eine **Lesesperre** (shared lock) hält andere Anwendungen vom Schreiben in die Tabelle ab. Lesesperren garantieren eine konsistente Datenbanktabelle während einer Abfolge von Leseoperationen, z.B. beim Erstellen eines Index. Es können mehrere Anwendungen gleichzeitig eine Lesesperre für eine Tabelle setzen.

Eine **Schreibsperre** (exclusive lock) verhindert jeden Zugriff einer anderen Anwendung auf die Tabelle. Schreibsperren werden für verschiedene Schreiboperationen benötigt, z.B. das Ändern der Tabellenstruktur. Nur jeweils eine Anwendung kann eine Schreibsperre für eine Tabelle definieren.

Eine TurboDB Tabelle kann auch exklusiv geöffnet werden. In diesem Modus wird auch die zugrundeliegende Datenbankdatei exklusiv geöffnet, was jeder anderen Anwendung unmöglich macht auf die Datei zuzugreifen. Eine Tabelle exklusiv zu öffnen ist einer Schreibsperre ähnlich, mit einigen wesentlichen Unterschieden:

- Schreibsperren werden von TurboDB verwaltet, der exklusive Modus dagegen vom Betriebssystem.
- Schreibsperren können auf eine bereits geöffnete Tabelle angewendet werden. Der exklusive Modus kann nur vor dem Öffnen der Tabelle gesetzt werden.

Anmerkung

Bei der Arbeit mit den TurboDB VCL Komponenten werden Sie bemerken, dass Lesesperren als write locks und Schreibsperren als total locks bezeichnet werden. Der Grund ist, dass die BDE die Sperrentypen durcheinander bringt und wir versuchen halbwegs kompatibel zu sein.

5.2.5.2 Satzsperren

TurboDB sperrt automatisch den Datensatz, wenn ein Anwender mit dem Editieren beginnt. Das verhindert, dass zwei Anwender denselben Datensatz gleichzeitig bearbeiten. Die Sperre wird aufgehoben wenn der Anwender die Änderung bestätigt (Post) oder Rückgängig macht (Cancel) oder wenn er die Tabelle schließt.

Wenn ein Anwender einen gesperrten Datensatz editieren möchte, wird ein Fehler zurückgegeben und der Anwender kann den Datensatz nicht ändern. Die meisten Komponenten-Bibliotheken (z.B VCL/CLX) werfen eine Exception falls der Fehler auftritt.

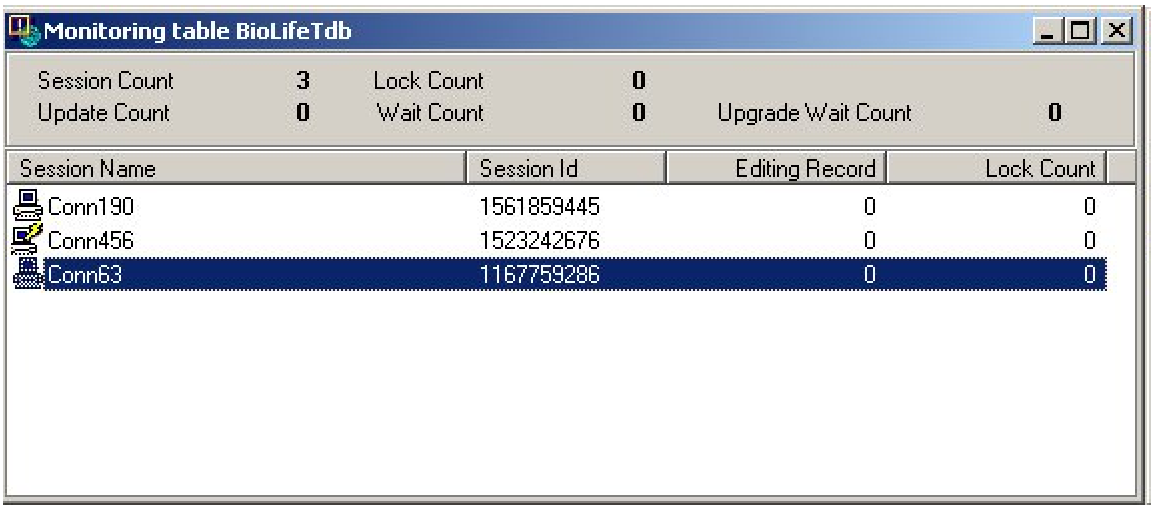
In TurboDB können Satzsperren neben Tabellensperren auftreten. Es ist möglich, dass ein Anwender beginnt einen Datensatz zu editieren und ein zweiter führt ein Update Statement auf die Tabelle aus, bevor der erste Anwender seine Änderung geschrieben hat. Das geht gut, solange das Update nicht den gesperrten Datensatz tangiert. Falls es das tut, wird das Update scheitern. In beiden Fällen kann der editierende Anwender die durchgeführten Änderungen schreiben.

Für einige Anwendungen mit sehr vielen Usern in einem losen Netzwerk (z.B. Web-Anwendungen), ist es keine gute Idee Datensatzsperren aufrechtzuerhalten während der Anwender den Satz editiert. In diesem Fall sollte ein unverbundenes Datenzugriffsmodell bevorzugt werden.

5.2.5.3 Anzeige der Tabellen-Nutzung

Da Sperren einen Einfluß auf die Performanz haben können und in manchen Fällen die korrekte Ausführung eines Programms verhindern (Deadlock), ist es manchmal hilfreich sich einen Überblick über die verschiedenen Anwendungen und User zu verschaffen, die auf eine Tabelle zugreifen. Dazu kann man einen Blick in die *.net Datei der Tabelle werfen, die alle nötigen Informationen beinhaltet.

TurboDB Viewer hat dazu den Menü-Befehl Tools/Table Usage, der den Netzwerkmonitor anzeigt.



Eine andere Mehode um den Inhalt der Net-Datei zu prüfen bietet das Kommandozeilen-Tool TdbLocks.exe, das für Linux und Windows im Download-Bereich der dataWeb Website bereitsteht.

Wenn Sie die Tabellen-Nutzung in Ihrer Anwendung überprüfen möchten, können Sie das, wenn Sie mit den VCL/CLX Komponenten von Delphi oder Kylix arbeiten, über die Methode TTdbTable.GetUsage erreichen. (ADO.NET arbeitet mit einem unverbundenen Datenmodel und kümmert sich daher recht wenig um Sessions die eine Tabelle verwenden.)

Egal auf welche Weise Sie einen Blick in die Net-Datei riskieren, die Information, die Sie vorfinden werden ist immer dieselbe:

Zuerst kommt die Tabellen-bezogene Information:

Session Count	Anzahl der momentanen Verbindungen, die diese Tabelle verwenden
Update Count	Anzahl der Number Änderungen seitdem die Tabelle von der ersten Verbindung geöffnet wurde
Lock Count	Anzahl der Sitzungen, die eine Lese-Sperre auf die Tabelle eingerichtet haben. Der Wert -1 bedeutet, dass eine Sitzung eine Schreib-Sperre auf die Tabelle hält
Wait Count	Anzahl der Sitzungen, die momentan darauf warten eine Sperre einzurichten
Upgrade Wait Count	Anzahl der Sitzungen, die momentan eine Lese-Sperre halten und darauf warten eine Schreib-Sperre einzurichten

Darauf folgt die Session-bezogene Information:

Session Name	Lesbarer Name einer Sitzung zur leichteren Identifikation. Der Name kann per Programmierung der Datenbank-Komponente festgelegt werden. Ist dies nicht der Fall, wird automatisch ein Name nach dem Schema ConnXXX erzeugt, wobei XXX eine Zufallzahl ist.
Session Id	Automatisch erzeugte, eindeutige Identifikation der Session
Editing Record	Der Datensatz, den die Sitzung momentan editiert (falls dies der Fall ist)
Lock Count	Korrespondiert mit Lock Count der Tabellen-bezogenen Information. Kann also die werte -1, 0 oder 1 haben.

5.2.6 Transaktionen

TurboDB unterstützt Transaktionen basierend auf einem zusätzlichen Speicherobjekt pro Tabelle, dem Redo-Log. Nach dem Start einer Transaktion werden alle folgenden Änderungen an den Datenbank-Tabellen in die Redo-Logs eingetragen. Ist die Transaktion mit Commit abgeschlossen, werden die Redo-Logs einfach gelöscht. Bei einem Rollback wird die im Redo-Log vorhandene Information verwendet um die Änderungen rückgängig zu machen. Das bedeutet, TurboDB folgt einem optimistischen Transaktionsschema: Das Committen einer Transaktion ist sehr schnell, während ein Rollback wesentlich aufwändiger ist.

Die während einer Transaktion geänderten Tabellen sind für andere Anwendungen gesperrt, bis die Transaktion abgeschlossen ist. Aus Gründen der Geschwindigkeit werden Tabellen, die während einer Transaktion gelesen werden, nicht gesperrt. Der TurboDB Transaktions-Level ist Read Commited.

Da TurboDB Anwendungen auf Dateiebene miteinander kommunizieren (d.h. ohne Datenbankserver), ist das Handling von Anwendungen, die während einer Transaktion sterben schwierig. TurboDB Engine erkennt, dass eine andere Anwendung gestorben ist und führt ein Rollback aus. Da in diesem Fall eine andere Anwendung die vom gestorbenen Client durchgeführten Änderungen rückgängig macht, nennen wir diesen Mechanismus *Hijacking*.

5.2.7 Optimierung

Wenn Sie das Gefühl haben, dass Ihre TurboDB Anwendung langsamer ist als sie sein sollte, kann es dafür viele Gründe geben. Prüfen Sie die folgenden Punkte und befolgen Sie die entsprechenden Ratschläge.

Abfragen gegen eine große Tabelle oder gegen mehrere Tabellen dauern sehr lange

Prinzipiell gibt es zwei Wege eine Abfrage zu beschleunigen: Die nötigtigen Indexe erstellen und/oder das Statement optimieren.

Filtern mit einer Tabellen Komponente (VCL) dauert sehr lange

Einen Index erstellen kann auch hier helfen. Eine andere Möglichkeit ist an Stelle des Filters das Range Feature zu verwenden.

Lokale Tabellen-Operationen sind sehr schnell, aber sobald sich eine zweite Anwendung mit der Datenbank verbindet wird alles sehr langsam.

Zuerst sollte geprüft werden ob es sich um ein ob es sich um ein Netzwerkproblem handelt. Da dateibasierter Zugriff extensiven Gebrauch der Netzwerkfunktionalität macht, kommt es öfters vor, dass schlechte Netzwerkperformanz erst nach Installation der TurboDB Anwendung bemerkt wird.

Das Netzwerk ist ok, editieren aber viele Anwender die Datenbank, dauert eine Operation sehr lange

Je mehr Anwender gleichzeitig auf einer Datenbank arbeiten, umso größer wird der Aufwand die Tabellensperren zu verwalten und es muss auch öfters gewartet werden bis auf die Tabellen zugegriffen werden kann. In diesem Fall ist der erste Schritt explizite Sperren für größere Operationen zu verwenden und dadurch die Anzahl der zu setzenden und wieder freizugebenden Sperren zu minimieren. Falls das nicht zum gewünschten Erfolg führt, kann noch der TurboDB Server verwendet werden.

Zu diesen eher spezifischen Tipps gibt es noch allgemeine Hinweise, die helfen können die Performanz zu steigern:

Den Flush Modus auf Fast setzen

Der Flush Mode bestimmt inwieweit die Datenbank Schreiboperationen puffert. Im Modus *Fast* ist das interne Puffer maximiert, ebenso die Geschwindigkeit. Es kann dann aber zu Datenverlust kommen, wenn die Anwendung abstürzt oder der Strom ausfällt.

Exklusiven Zugriff verwenden falls möglich

Falls die Anwendung nur für einen Anwender gedacht ist, sollte die Datenbank im Exklusivemodus betrieben werden. Damit entfällt der gesamte Aufwand der Multi-User Verwaltung.

5.2.7.1 Netzwerk Durchsatz und Wartezeit

Die Performanz des Netzwerk ist äußerst kritisch für die Performanz der TurboDB Anwendung, speziell dann, wenn mehrere Anwendungen auf eine Datenbank zugreifen. Probleme mit der Netzwerkkonfiguration sind der häufigste Grund für schlechten Daten-Durchsatz und sollten daher immer als erstes geprüft werden. Ein Netzwerkproblem kann vorliegen, wenn sich die Datenbank auf einem anderen Rechner befindet als die Anwendung und diese generell sehr langsam ist oder langsam wird, wenn ein zweiter Anwender auf die Datenbank zugreift.

Da Netzwerkprobleme oft sehr schwer zu identifizieren sind, bietet dataWeb das Freeware Programm *NetTest* an, das die für Datenbank-Anwendungen wichtigen Parameter des Netzwerkes misst. Sie können das Programm jederzeit beim dataWeb Support anfordern. In fünf Minuten können Sie dann herausfinden, ob Ihr Netzwerk der Schuldige ist oder nicht.

Falls das Netzwerk nicht schuld an dem Performanz-Problem ist, sind da mehrere Punkte die geprüft werden müssen:

- Prüfen Sie ob ein Anti-Viren Programm oder eine andere Software dieser Art auf TurboDB Dateien zugreift. Diese Programme tendieren dazu Tabellen-Dateien, die sich ja sehr oft ändern, nach jeder Änderung zu überprüfen und dabei den Datenbankzugriff zu verlangsamen oder ganz zu unterbinden. Anti-Viren Software sollte so konfiguriert sein, dass TurboDB Tabellen von der Überprüfung ausgeschlossen sind. Das führt zu keinen Sicherheitsloch, das es sich dabei nicht um ausführbare Dateien handelt.
- Prüfen Sie ob es für Ihre Netzwerkkarte einen aktuelleren Treiber gibt oder ob der Adapter defekt ist
- Es gibt ein bekanntes Problem mit SMB Signing, falls ein Windows XP Rechner auf einen Windows 200 Domain Controller zugreift. Weitere Informationen und die Lösung finden Sie in der Microsoft Knowledge Base, Artikel 321098.
- Ein Hub zwischen den Datenbank Clients und dem Datenbank Fileserver blockiert manchmal den Zugriff, falls der Netzwerk-Traffik zu hoch ist. In diesem Fall sollte der Hub durch einen guten Switch ersetzt werden.

5.2.7.2 Sekundäre Indexe

Zusätzliche Indexe für Tabellen können Abfragen und Filter um Größenordnungen beschleunigen. Betrachten wir eine einfache Abfrage wie:

```
select * from Customers where State = 'NJ'
```

oder der vergleichbaren Filterbedingung

```
State = 'NJ'
```

Ohne Index muss TurboDB jeden Datensatz der Tabelle prüfen um diejenigen herauszufinden, die der Bedingung entsprechen. Und obwohl TurboDB wiederholte Lesevorgänge optimiert, kann die Operation für große Tabellen (einige Millionen Datensätze) einige Minuten dauern.

Wenn dagegen ein Index vorhanden ist, der mit der State-Spalte beginnt, kann die Ergebnismenge augenblicklich geliefert werden, da TurboDB die zutreffenden Datensätze direkt aussortieren kann.

Das gilt auch für Joins, eine zusätzlicher Index kann Wunder bewirken:

```
select * from Customers, Orders where Orders.CustNo = Customers.No
```

oder die Entsprechung

```
select * from Customers join Orders on Orders.CustNo = Customers.No
```

Auch hier wird ein Index über *Orders.CustNo* oder *Customers.No* die Abfrage beträchtlich beschleunigen. Ja nachdem welcher Index existiert wird TurboDB die Abfrage so abarbeiten, dass der Index verwendet werden kann. Da jedoch die *Orders*-Tabelle wahrscheinlich wesentlich größer ist als die *Customer*-Tabelle (die durchschnittliche Anzahl an Bestellungen pro Kunde ist hoffentlich größer als eins), wird ein Index über das Feld *Orders.CustNo* hinsichtlich der Geschwindigkeit mehr bringen als ein Index über *Customer.No* (Letztere wird meistens sowieso existieren, da die Kundennummer der Primärschlüssel der Kunden-Tabelle sein wird).

Der Nachteil an Indexen ist, dass die Wartung während Änderungen (Editieren, Einfügen, Löschen) wiederum Zeit bedarf. Es muss daher die Geschwindigkeit der gesamten Anwendung im Auge behalten werden. In vielen Anwendungen sind Abfragen wesentlich häufiger als Änderungen, daher zahlen sich Indexe für die entscheidenden Anwendungsfälle meistens aus.

5.2.7.3 TurboSQL Statements

Einige Grundsätze für schnelle TurboSQL Abfragen:

Where und Having Klauseln sollten mit einfachen, mit and verknüpften Bedingungen beginnen

Falls es logisch machbar ist sollte die Bedingung so angeordnet sein:

```
A.a = B.a and C.b > X and (...)
```

D.h. mit einfachen Vergleichen beginnen, die für die gesamte Suchbedingung erfüllt sein müssen. Diese simplen Vergleiche sind am besten zur Optimierung geeignet. Der Optimierer versucht derartige Strukturen automatisch für die gegebene Suchbedingung zu erstellen, kann aber in manchen Fällen nicht schlau genug dazu sein.

Spalten-Bezeichner in Vergleichen separieren

Schreiben Sie

```
A.a > 2 * :ParamValue,
```

wird das besser optimiert als

```
A.a/2 > :ParamValue.
```

Entscheidend ist hier, dass die Referenz der Spalte *A.a* alleine auf der linken Seite des Vergleichs steht.

like statt Upper verwenden

Die Bedingung

```
A.a like 'USA'
```

kann optimiert werden

```
Upper(A.a) = 'USA'
```

dagegen nicht.

Left Outer Joins statt Right Outer Joins verwenden

Die Implementierung von Joins bevorzugt Left Outer Joins. Immer wenn es in einer Anwendung möglich ist, sollte

```
B left outer join A on B.a = A.a
```

verwendet werden statt

```
A right outer join B on A.a = B.a.
```

Das kann die Abfrage merklich beschleunigen. Der Optimierer macht die Konversion nicht

automatisch, da sonst keine Möglichkeit bestünde, die Abfrage von Hand zu optimieren.

Reihenfolge der Tabellen in der From Klausel ändern

Diese Reihenfolge kann eine erhebliche Auswirkung auf die Geschwindigkeit der Abfrage haben. Falls Sie den Eindruck haben Ihre Abfrage könnte schneller sein, versuchen Sie die Reihenfolge der Tabellen zu variieren.

```
select * from A, B, C
where ...
```

kann viel schneller sein als

```
select * from C, B, A
where ...
```

Normalerweise wird der Optimierer die Reihenfolge der Tabellen, die nicht Teil eines Joins sind, optimal zu wählen, manchmal ist die Assistenz eines Programmierers hilfreich.

5.2.8 Fehlerbehandlung

Die Datenbank Engine meldet aufgetretene Fehler in Form von Exceptions.

5.2.8.1 Fehlerbehandlung in TurboDB Native

Es besteht immer die Möglichkeit, dass die Ausführung eines TurboDB Befehls fehlt schlägt. Die Ursachen dafür gehen von der Verwendung falscher Argumente, über Verstöße gegen die Integrität der Datenbank und Tabellensperren, bis hin zu Hardware-Fehlern und vollen Speichermedien. Tritt ein derartiges Problem auf, löst TurboDB eine Exception aus. Die genaue Klasse dieser Ausnahme hängt von der verwendeten Bibliothek ab, unabhängig davon umfasst die Fehlerinformation vier wesentliche Bestandteile:

Exception Klasse

Es gibt eine Basisklasse für alle TurboDB Exceptions und einige abgeleitete Klassen für die diversen Fehlerkategorien. Abhängig von der verwndeten Bibliothek können auch Standard-Exceptions der Bobliothek auftreten (wie *Tabelle muss geöffnet sein* in der VCL Bibliothek oder *Falscher Typ* in .NET).

Codes für die Fehlerbeschreibung

Die Basisklasse der TurboDB Exceptions verfügt über einen Fehlercode, der die fehlgeschlagene Operation bezeichnet. Im folgenden Abschnitt findet sich eine Liste der Codes für die Fehlerbeschreibung und ihre Bedeutung.

Codes für die Fehlerursache

Die Basisklasse der TurboDB Exceptions verfügt auch über eine Eigenschaft, die den Grund des Fehlers angibt. Während beispielsweise die Fehlerbeschreibung lautet Index kann nicht erstellt werden, kann der Code für die Fehlerursache entweder *Indexdatei existiert bereits* oder *Index bereits geöffnet* bedeuten. Der übernächste Abschnitt listet die Codes für die Fehlerursache und ihre Bedeutung.

Wichtige Regel zur Fehlerbehandlung

Gehen Sie nie davon aus den Grund für einen Fehler an einer bestimmten Code-Stelle zu kennen. Bei Verwendung einer Bibliothek für verbundenen Zugriff (z.B. VCL Tabellen oder das RecordSet unter .NET) darf nie folgendes gemacht werden (Es handelt sich hier um Pseudo-Code, der von allen Programmieren zu verstehen ist):

```
try
  ConnectedComponent.Post
catch
  // Post schlägt fehl, weil der Anwender ungültige Daten eingegeben hat
  ShowMessage('Die eingegebenen Daten sind ungültig, bitte korrigieren.');
end;
```

Das dürfen Sie so nicht machen, denn es kann and dieser Stelle viele andere Gründe für eine Exception geben und einige davon sollten nicht auf die leichte Schulter genommen werden: Auf

einer Ebene kann die Kapazität des Speichermediums oder der Tabelle erschöpft sein, ein Index kann kaputt sein (wegen eines vorhergehenden Crashs), eine Datenbankdatei kann blockiert sein (beispielsweise durch einen Virenscanner). Auf einer anderen Ebene kann die Anwendung einen Fehler haben, die Komponente wurde nicht erzeugt, die Connection nicht geöffnet, der Datensatz nicht in den Editier- oder Anfügemodus versetzt, usw. In all diesen Fällen wird die gezeigte Fehlerbehandlung nicht nur Ihre Anwender behindern, sondern kann auch Auswirkungen auf die Integrität der Datenbank haben da man nicht in der Lage ist auf die eventuell fatalen Zustand zu reagieren. Eine korrekte Implementierung für obiges Beispiel sieht so aus (wieder in Pseudo-Code unter Verwendung verständlicher Bezeichner):

```
try
  ConnectedComponent.Post
catch(TurboDBError Exc)
  if Exc.ErrorCode = TdbErr_InvalidData then
    // Post schlägt fehl, weil der Anwender ungültige Daten eingegeben hat
    ShowMessage('The data you entered is invalid, please correct it.');
  else throw;
end;
```

Jetzt wird die Meldung wirklich nur im vorhergesehenen Fall ausgegeben. Für alle anderen Ereignisse wird die Exception auf einer höheren Ebene behandelt oder zum Programmabbruch führen, was immer noch besser ist als den Fehler in einigen Fällen zu unterdrücken.

Obwohl sich diese Diskussion auf das Modell des sequentiellen Datenzugriffs konzentriert, ist die Essenz genauso auf den SQL-basierten Datenzugriff anwendbar. Obwohl Sie durch das Ausführen von SQL-Anweisungen keinen Einfluss auf die Integrität der Datenbank nehmen können, kann die Annahme den der Grund für eine Ausnahme im Voraus zu kennen, immer noch zu irreführenden Fehlermeldungen für den Benutzer und zu seltsamen Verhalten der Anwendung führen.

5.2.8.1.1 Codes für die Fehlerbeschreibung

Die Object Pascal Konstanten für diese Fehler Codes sind in der Unit *TdbTypes* definiert. Viele dieser Fehler haben einen zusätzlichen Reason Code.

Constant	Value	Message
TdbErr_Ok	0	Kein Fehler
TdbErr_RelationNotFound	-1	Kann Relation nicht öffnen
TdbErr_OutOfMemory	-2	Zuwenig Speicher
TdbErr_ModuleNotFound	-3	Kann Modul nicht öffnen
TdbErr_InvalidRecord	-5	Ungültiger Datensatz
TdbErr_InvalidExpression	-7	Ungültiger Ausdruck
TdbErr_InvalidField	-10	Unbekannte Tabellenspalte
TdbErr_RecNotFound	-11	Datensatz existiert nicht
TdbErr_WrongPassword	-12	Ungültiges Passwort
TdbErr_CreateFailed	-14	Relation kann nicht erzeugt werden
TdbErr_InvalidAccess	-15	Ungültiger Zugriff
TdbErr_InvalidIndex	-16	Der angegebene Index existiert nicht
TdbErr_InvalidRelation	-17	Ungültiger Relation Handle
TdbErr_InvalidIndexSpec	-18	Ungültige Index Definition
TdbErr_InvalidFunction	-19	Die Funktion kann in diesem Kontext nicht aufgerufen werden
TdbErr_DeleteFailed	-20	Kann Datensatz nicht löschen

TdbErr_RestructureFailed	-21	Restrukturierung der Relation fehlgeschlagen
TdbErr_NoAdlRltn	-22	Die Relation verfügt über keine RecordId Spalte
TdbErr_InvalidType	-23	Spalte hat nicht den korrekten Datentyp
TdbErr_WriteFailed	-25	Kann Datensatz nicht in die Tabelle schreiben
TdbErr_DuplicateRecord	-26	Datensatz verstößt gegen eindeutigen index
TdbErr_IndexInUse	-27	Der Index wird noch benutzt
TdbErr_RltnNotWritable	-30	Die Relation ist nur zum Lesen geöffnet
TdbErr_SharingConflict	-31	Der Datensatz ist von einer anderen Anwendung gesperrt
TdbErr_IndexNotAvailable	-32	Index nicht verfügbar
TdbErr_RecordIsEmpty	-33	Datensatz ist Null
TdbErr_RecordIsUnchanged	-38	Der Datensatz wurde nicht geändert
TdbErr_NoOdbcSupport	-39	Diese Edition unterstützt ODBC nicht
TdbErr_ProcNotFound	-40	Die Prozedur existiert nicht
TdbErr_InvalidModule	-41	Das Modul existiert nicht
TdbErr_DuplicateProc	-42	Die Prozedur existiert bereits
TdbErr_InvalidIdentifier	-43	Der Bezeichner ist kein gültiger TurboDB Bezeichner
TdbErr_InvalidCondition	-44	Ungültige Suchbedingung
TdbErr_RltnAlreadyOpen	-45	Die Relation ist bereits geöffnet
TdbErr_SqlFailure	-46	Fehler beim Ausführen des SQL-Befehls
TdbErr_RltnAlreadyLocked	-47	Die Relation ist bereits gesperrt
TdbErr_IndexDeleteFailed	-48	Index kann nicht gelöscht werden
TdbErr_RltnCloseFailed	-49	Relation kann nicht geschlossen werden
TdbErr_InvalidEquateJoin	-50	Fehlerhafter Equate Join
TdbErr_RltnDeleteFailed	-51	Relation kann nicht gelöscht werden
TdbErr_RelTableNoFound	-52	Relationstabelle (*.rel) nicht gefunden
TdbErr_CreateSessionFailed	-53	Session konnte nicht angelegt werden
TdbErr_CannotCreateFile	-54	Datei konnte nicht erzeugt werden
TdbErr_TransferRunning	-55	Es läuft schon ein Blob-Transfer für diesen Cursor
TdbErr_NoActiveTransfer	-56	Es läuft noch kein Blob-Transfer für diesen Cursor
TdbErr_NotInEditMode	-57	Datensatz ist nicht im Editiermodus
TdbErr_DatabaseNotFound	-58	Datenbankverzeichnis oder Datenbankdatei nicht gefunden
TdbErr_SqlSyntaxError	-59	SQL-Befehl entspricht nicht der Turbo SQL Syntax
TdbErr_CannotExecuteCommand	-60	Konnte SQL-Befehl nicht ausführen
TdbErr_EngineError	-100	Interner Fehler
TdbErr_InvalidTable	-101	Der Handle der Tabelle ist ungültig
TdbErr_InvalidStmt	-104	Der Statement Handle ist ungültig

TdbErr_ReadFailed	-201	Fehler beim Lesen aus Tabelle
TdbErr_Unspecified	-203	Unspezifizierter Fehler
TdbErr_ExportFailed	-204	Export fehlgeschlagen
TdbErr_ImportFailed	-205	Import fehlgeschlagen

5.2.8.1.2 Codes für die Fehlerursache

Falls der Reason Code ungleich null ist, beschreibt er die Ursache des Fehlers.

Code	Value	Description
fcCannotOpenFile	1	Datei oder Speicherobjekt kann nicht geöffnet werden
fcReadError	2	Fehler beim Lesen aus Datei oder Speicherobjekt
fcWriteError	3	Fehler beim Schreiben in Datei oder Speicherobjekt
fcIndexCreating	4	Fehler beim Erstellen eines Index
fcIndexDeleting	5	Fehler beim Löschen eines Index
fcIndexNotFound	6	Index nicht vorhanden
fcInvalidCondition	7	Fehler in Suchbedingung
fcImportError	8	Import-Fehler
fcOutputFormatError	9	Fehler in Ausgabeformat
fcCloseError	10	Fehler beim Schließen der Datei
fcIndexTooLarge	11	Größe des Index-Schlüssels darf 32KB (512 Bytes für Tabellen-Level 3 und kleiner) nicht überschreiten
fcOpeningIndexError	12	Fehler beim Öffnen des Index
fcIndexDoesNotFit	13	Index paßt nicht zur Tabelle
fcNoDataField	14	Ein Datenfeld wird im Ausdruck erwartet ist aber nicht vorhanden
fcNoFileInDir	18	Ein ungültiger Handle wurde an TurboPL übergeben
fcRecordNotFound	19	Datensatz nicht gefunden
fcModuleNotFound	20	Modul nicht gefunden
fcIndexAlreadyExists	22	Indexdatei existiert bereits
fcIndexAlreadyOpen	23	Index bereits geöffnet
fcSyntaxError	25	Syntax-Fehler
fcInvalidFileName	26	Illegaler Dateiname
fcIndexDescDamaged	28	Indexbeschreibung zerstört
fcWrongProgramFileVersion	29	Falsche Programmversion
fcDuplicateEntry	30	Doppelter Eintrag
fcCannotOpenMemoFile	31	Memo-Datei kann nicht geöffnet werden
fcMemoNotAllowedHere	32	Memo hier nicht erlaubt
fcWritingMemoFileError	33	Fehler beim Schreiben in die Memo-Datei
fcIllegalOperation	34	Anwendung hat nicht die Zugriffsrechte für diese Operation

fcTableAlreadyOpen	35	Tabelle ist bereits geöffnet
fcFileNotFound	37	Datei nicht vorhanden oder Zugriff verweigert
fcWrongKey	38	Das Passwort ist nicht korrekt
fcExpressionIncomplete	41	Ausdruck unvollständig
fcOperatorNotAllowedForOperand	42	Operator paßt nicht zu Operand
fcRealOverflow	43	Real-Überlauf
fcTypeMismatch	44	Typen stimmen nicht überein
fcIllegalCharacter	45	Zeichen an dieser Position im Ausdruck ungültig
fcInvalidNumber	46	Die Zeichenkette repräsentiert keine Zahl.
fcLogicalOperandMissing	48	Ein logischer Operand erwartet aber nicht vorhanden
fcIllegalOperand	49	Ungültiger Wert für Operand
fcUnknownIdentifier	50	Unbekannter Bezeichner
fcArrayVariableExpected	51	Array Variable erwartet
fcUnknownError	52	Unbekannter Fehler
fcTooManyVariables	53	Es wurde versucht mehr als 2048 Variablen zu definieren
fcEqualMissing	54	"=" im Ausdruck erwartet aber nicht vorhanden
fcNumberExpected	55	Zahl im Ausdruck erwartet aber nicht vorhanden
fcUnexpectedToken	56	Ein spezielles Merkmal im Ausdruck erwartet aber nicht vorhanden
fcInvalidColumnNumber	57	Spaltenzahl in der Quelltabelle entspricht nicht Spaltenzahl in der Zieltabelle
fcTableNameExpected	58	Tabellenname erwartet
fcTooManyColumns	59	Zu viele Tabellenspalten
fcExpressionTooComplex	60	Ausdruck zu komplex
fcTooManyIndexLevels	61	Es wurde versucht mehr als 16 Indexstufen für eine Tabelle mit Tabellen-Level < 3 zu definieren
fcNotADLTable	64	Die Tabelle hat keine AutoInc-Spalte und kann nicht als Master-Tabelle für Link- und Relationsfelder verwendet werden.
fcADLTableNotFound	65	Die über Link- oder Relationsfeld verknüpfte Tabelle kann nicht gefunden werden.
fcNoRestructureAccess	66	Das Recht die Tabellenstruktur zu ändern ist nicht vorhanden
fcTooManyIndexes	67	Die Zahl der möglichen Indexe pro Tabelle, 50 bzw. 15 für Tabellen-Level 3 und niedriger, ist überschritten
fcInvalidRange	70	Bereich nicht festgelegt
fcDuplicateIdentifier	71	Bezeichner doppelt definiert
fcTableFileAlreadyExists	72	Tabellen-Datei existiert bereits
fcTooManyLinkFields	73	Zu viele Relations-Felder

fcUpToFifteenDigitsAllowed	74	Nur 15 Stellen erlaubt
fcLineTooLong	75	Modulzeile hat mehr als 255 Buchstaben
fcMoreThanOneAutoIncField	76	Eine Tabelle kann nur eine AutoInc-Spalte haben
fcRightMarginTooSmall	79	Zu wenig Platz am rechten Rand
fcRangeOverflow	81	Bereichsüberlauf
fcIllegalCommand	82	Das TurboPL Kommando ist hier nicht erlaubt
fcInvalidAreaSequence	86	Bericht-Bereiche haben die falsche Reihenfolge
fcTableNotOpen	88	Tabelle nicht geöffnet
fcVariableExpected	89	Eine Variablenname erwartet aber nicht vorhanden
fcIndexWriting	90	Fehler beim Schreiben des Index
fcIndexReading	91	Fehler beim Lesen des Index
fcEndOfSubreportNotFound	92	Subreport begonnen aber endsub nicht vorhanden.
fcKeywordNotAllowed	93	Das Schlüsselwort ist im aktuellen Bereich nicht erlaubt
fcTooManyTables	94	Es können maximal 254 Tabellen pro Session geöffnet werden (62 für Tabellen-Level 3 und niedriger)
fcIndexDamaged	95	Der Index ist beschädigt
fcUntilExpected	96	Repeat Schleife begonnen aber until nicht vorhanden
fcEndExpected	97	if oder while Block begonnen aber end nicht vorhanden
fcInvalidIndexSpec	98	Ungültige Index Definition
fcOutOfMemory	99	Speicher reicht nicht aus
fcDemoVersion	100	Demoversion erlaubt nicht mehr Datensätze
fcLocalProceduresNotPermitted	101	TurboPL unterstützt keine verschachtelten
fcEndProcExpected	102	Eine Prozedur wurde begonnen aber endproc nicht vorhanden
fcTableInUse	103	Die Operation kann nicht abgeschlossen werden, weil die Tabelle von einer anderen Anwendung/Session verwendet wird
fcTableIsLocked	104	Die Tabelle ist von einer anderen Anwendung/Session gesperrt
fcRecordEdited	105	Der Datensatz ist von einer anderen Anwendung/Session gesperrt
fcErrorInLogin	106	Die Tabelle kann nicht geöffnet werden, weil die Verwaltung der Netzdateien nicht erfolgreich war
fcInvalidConnectionId	107	Ungültige Connection Id
fcUnknownLockError	109	Unbekannter Netzwerkfehler
fcLockingError	110	Dateisperren werden vom Betriebssystem nicht ausreichend unterstützt
fcCannotOpenBlob	111	Blob Datei kann nicht geöffnet werden
fcMemoDamaged	112	Memo Datei ist beschädigt
fcBlobDamaged	113	Blob Datei ist beschädigt

fcUserAbort	114	Die Operation wurde vom Benutzer abgebrochen
fcNoWriteAccess	115	Kein Schreibrecht auf Datei
fcIndexInUse	116	Die Operation kann nicht abgeschlossen werden, weil der Index von einer anderen Anwendung/Session benutzt wird
fcUnsupportedTableFeature	117	Das Schema der Tabellen ist ungültig, weil es ein Merkmal verwendet, das nur von einem höheren Tabellen-Level unterstützt wird
fcErrorInExecutionExternalProcedure	118	Fatal error while executing external procedure
fcExternalProcedureNotFound	119	Fataler Fehler beim Aufruf der externen Prozedur
fcNoOdbcSupport	120	ODBC konnte nicht initialisiert werden
fcCannotOpenODBCDataSource	121	Die ODBC-Datenquelle kann nicht geöffnet werden
fcErrorInQuery	122	Bei der SQL-Abfrage ist ein Fehler aufgetreten
fcException	123	Ausnahme in TurboDB Engine
fcOldDatVersion	124	Veraltete Version der Tabellen-Datei
fcCannotAssignToConst	125	Einer Konstanten kann nichts zugewiesen werden
fcObjectExpected	126	Links von . muss ein Objekt stehen
fcArrayTooLarge	127	Array hat zuviele Elemente (bis zu 2 GB möglich)
fcInvalidFullTextSearch	128	Die Volltext-Suchbedingung enthält einen Fehler
fcUnknownDBaseFormat	129	Die Version der dBase-Datei ist unbekannt
fcSharingViolation	130	Die Datei wird noch von einem anderen Anwender benutzt
fcUnknownClass	132	Unbekannte Klasse
fcInvalidObject	133	Ungültige Objekt-Referenz
fcTableCorrupt	134	Dateikopf der Tabelle ist beschädigt
fcErrorInUICall	135	Fehler beim Aufruf einer Bibliotheks-Routine
fcInvalidMember	136	Unbekanntes Klassen-Element
fcInvalidDate	137	Die Zeichenkette repräsentiert kein gültiges Datum
fcLangDriverNotFound	138	Sprachtreiber wurde nicht gefunden oder entspricht nicht der Spezifikation
fcInvalidAggregate	139	Das Argument einer Aggregations-Funktion muss nummerisch sein
fcInvalidTime	140	keine gültige Zeitangabe
fcNoCreatePermission	141	Kein Recht, Datei zu erstellen
fcNoReadPermission	142	Kein Recht, Datei zu lesen
fcFieldSizeIsZero	143	Feld hat die Größe Null
fcUnknownFieldType	144	Unbekannter Feldtyp
fcIndicationMissing	145	AutoInc-Feld hat keine Anzeigeinformation

fcInvalidJoin	146	Ungültiger Join
fcDifferentLangDriver	147	Alle Tabellen einer Sitzung müssen denselben Sprachtreiber benutzen
fcInvalidStrValue	148	kein gültiger Wert fr das Feld
fcNoIndexPermission	149	Kein Recht, einen Index zu erstellen
fcValueListTooLong	150	Zu viele Werte in der Liste
fcSingleTableJoin	151	Equate join hat genau eine Tabelle auf jeder Seite des =
fcNotDBaseCompatible	152	DBase III-Dateien können diese Datenstruktur nicht aufnehmen
fcExternalTableError	153	Fehler in externer Tabelle
fcRelTableNotFound	154	Mindestens eine Relationstabelle konnte nicht geöffnet werden
fcDeletionNotComplete	155	Eine oder mehrere Dateien konnten nicht gelöscht werden
fcBlobFileTooLarge	156	Die Memo/Blob Datei überschreitet die maximale Größe
fcNoAutoIncModify	157	AutoInc Feld kann nicht geändert werden
fcInvalidDateTime	158	Ungültiger Zeitstempel
fcLangDriverNotSupported	159	Sprachtreiber wird vom Betriessystem nicht unterstützt
fcNoWritePermission	160	Schreibrecht für Datei nicht vorhanden
fcExpressionHasNoValue	161	Der Ausdruck hat keinen Wert, auch nicht Null
fcWrongFieldType	162	Der Spaltentyp ist nicht wie erwartet
fcNextExpected	163	for-loop wurde begonnen aber next nicht vorhanden
fcUnknownType	164	Der Typname ist unbekannt
fcCOMError	165	Fehler beim Aufruf eines COM Interface
fcCOMError	166	Module verwendet sich direkt oder indirekt selbst
fcConstraintViolated	167	Datensatz verletzt eine Gültigkeitsbedingung
fcBlobFieldExpected	168	Spaltenreferenz auf Blob erwartet aber nicht vorhanden
fcMemoFieldExpected	169	Spaltenreferenz auf Memo erwartet aber nicht vorhanden
fcLocksPresent	170	Transaktion kann nicht gestartet werden solange Sperren auf der Tabelle sind
fcCapacityLimit	171	Einer der Tabellenindexe hat seine maximale Kapazität erreicht und muss neu erstellt werden
fcIncompleteGroupBy	172	Group By Klausel nicht vollständig
fcInvalidColumnName	173	Zeichenkette ist kein gültiger Spaltenbezeichner
fcIdentifierAmbigous	174	Bezeichner nicht eindeutig
fcTableStillBusy	175	Tabelle wird noch von dieser Session verwendet
fcInvalidIndexName	176	Zeichenkette ist kein gültiger Indexname
fcNotEnoughArguments	177	Prozedur hat nicht genügend Parameter
fcDivisionByZero	178	Division durch Null aufgetreten

fcNoParentRowFound	179	Fremdschlüsselbedingung kann nicht erfüllt werden
fcIntegrityViolated	180	Operation würde Fremdschlüsselbedingung verletzen
fcRecordNotEditing	181	Datensatz ist gesperrt
fcOutOfTable	182	Position des Cursors ist außerhalb der Tabelle
fcTransactionRunning	183	Es läuft bereits eine Transaktion
fcParentTableNotFound	184	Parent Tabelle für eine Fremdschlüsselbedingung ncht gefunden
fcIncompatibleLockFile	185	Anwendungen inkompatibler TurboDB Versionen verwenden die Tabelle

5.2.9 Verschiedenes

Datenbank-Dateien beschreibt welche physikalischen Datenbankdateien TurboDB kennt und wozu sie gut sind.

Datensicherheit behandelt die verschiedenen Methoden um Ihre Daten zu schützen.

Sprachunabhängigkeit skizziert den Weg wie Sie Ihre TurboDB Anwendung an spezielle Gebietsschemata anpassen.

5.2.9.1 Datenbank-Dateien

Hier werden die Dateien beschrieben, die zu einer TurboDB Datenbank gehören. Sie werden nach Ihrer Erweiterung unterschieden. Wenn Sie mit einer SingleFile-Datenbank arbeiten, können Sie diese Erweiterungen nicht direkt sehen. Verwenden Sie dataWeb Compound File Explorer, dann sehen Sie, dass die Datenbank-Datei Speicherobjekte mit denselben Erweiterungen enthält.

dat und rel Dateien

Beinhalten die Datenbanktabellen, also die Datensätze. Rel Dateien sind spezielle Datenbanktabellen die automatisch erzeugt werden um n:m Relationen zu implementieren. Diese Dateien sind mit einer Anwendung weiterzugeben.

mmo und blb Dateien

Das sind die Memo und Blob Dateien, die für jede Tabelle existieren, die über mindestens ein Memo bzw. mindestens ein Blob Feld verfügt. Eine solche Datei beinhaltet alle die Daten aller Memo oder Blob Felder der Tabelle. Memo oder Blob Dateien müssen mit einer Anwendung weitergegeben werden.

ind Dateien

Anwederspezifische Indexe. Jede ind Datei enthält einen Index. Diese Dateien sind mit einer Anwendung weiterzugeben.

id und inr Dateien

Automatisch erstellte Indexe für den Primärindex, falls definiert. Die id Datei ist ein Index über die Standard Sortierordnung and die inr Datei ist der Index über das AutoInc Feld. Diese Dateien sind mit einer Anwendung weiterzugeben.

net, rnt, mov und rmv Dateien

Das sind Netzwerkverwaltungsdateien und existieren für jede Tabelle, die im Shared Modus geöffnet wurde. Geben Sie diese Dateien nicht mit Ihrer Anwendung zum Kunden. Sie enthalten nur dynamisch erzeugte Information. Falls die Anwendung abstürzt oder während dem Debuggen abgebrochen wird, bleiben diese Dateien auf der Festplatte zurück und können beim erneuten Start der Anwendung zu Fehlermeldung, wie "Tabelle wird von einer anderen Anwendung benutzt", führen. Stellen Sie in diesem Fall sicher, dass keine Anwendung die betreffende Tabelle im Zugriff hat und löschen Sie dies *.net und *.mov Dateien von Ihrer Festplatte.

tra and rtr Dateien

Diese DAteien sind die Redo-Logs für Transaktionen. Während einer Transaktion, gibt es eine tra Datei (rtr für Relations Tabellen) für jede Tabelle, die während einer Transaktion geändert wird. Nach Ende der Transaction werden diese Dateien gelöscht. Falls eine dieser Dateien ohne

laufende Transaktion in der Datenbank zu sehen ist bedeutet dies, dass eine Anwendung während einer Transaktion gestorben ist. Löschen Sie die Redo-Logs nicht. Die Anwendung, die als nächstes die Tabellen verwendet wird ein Rollback für die unterbrochene Transaktion ausführen, um die Integrität der Datenbank zu gewährleisten.

Temporäre Tabellen und Indexe

Temporäre Tabllen haben Zufallsnamen wie jzbgopqw.dat und die temporären Indexe heißen entsprechend. Diese Dateien werden für gewöhnlich im temporären Windows Verzeichnis oder im Home Verzeichnis unter Linux abgespeichert. Sie können aber auch andere Verzeichnisse bestimmen indem Sie die Eigenschaft PrivateDir der TTdbDatabase Komponente setzen.

tdbd Dateien

tdbd steht für TurboDB Datenbank. Eine solche Datei enthält alle Tabellen und Indexe, die zu einer Datenbank gehören, wenn die Datenbank als Single-File-Datenbank angelegt wurde. In diesem Fall gibt es keine dat, mmo, blb, rel, id, in? und ind-Dateien, weil die entsprechenden Daten alle innerhalb der tdbd-Datei abgelegt sind.

5.2.9.2 Datensicherheit

Normalerweise kann Ihre TurboDB Datenbanktabelle von jeder Person gelesen werden, die Zugriff auf die Datei hat und die über ein Werkzeug verfügt, mit dem TurboDB Dateien geöffnet werden können. Um das zu verhindern, können Sie Ihre Tabellen mit einem Passwort schützen. Alle TurboDB Tools verlangen Sie dieses Passwort und werden den Inhalt der Tabelle nicht anzeigen, bis der Anwender das richtige Passwort eingegeben hat.

Obwohl das in vielen Fällen bereits ausreichend sein kann, ist es kein wirklicher Schutz Ihrer Daten. Wie auch andere dateibasierte Datenbanken (z.B, Access, dBase, Paradox) speichert TurboDB die Daten direkt in den Datenbankdateien. Das bedeutet, dass der Inhalt mit einem Binäreditor oder sogar einem beliebigen Texteditor eingesehen werden kann. Dies trifft auch zu, wenn Sie Ihre Tabelle mit einem Passwort versehen, da der Passwortschutz nicht die Art verändert, mit der die Daten in der Datei abgelegt werden. Falls Sie Ihre Daten effektiv vor unautorisierten Blicken schützen wollen, bietet TurboDB verschiedene Verschlüsselungs-Algprithmen an, die jeden Datensatz verschlüsseln bevor er in die Tabellen-Datei geschrieben wird.

Die klassische TurboDB-Verschlüsselung basiert auf einem 32 Bit Schlüssel. Wie Sie sicher wissen ist ein 32 Bit Schlüssel nicht sicher genug für Banking oder andere Hochsicherheits-Angelegenheiten. Für die meisten Anforderungen wird dieser Sicherheitslevel aber völlig ausreichend sein und ein kurzer Schlüssel sorgt für schnelle Datenbankaktionen.

Wenn Sie sicherere Verschlüsselung für Ihre Daten benötigen, können Sie einen der starken Verschlüsselungalgorithmen verwenden, die in TurboDB angeboten werden. Diese Algorithmen schützen Ihre Daten von jedem, der nicht den Schlüssel kennt. Zum derzeitigem Stand der Verschlüsselungtechnologie, können diese Chiffre sogar mit hoch entwickelten Dekodierungalgorithmen und Computerhardware nicht entschlüsselt werden.

Die Verschlüsselungmethode kann auf Datenbankebene (für verwaltete Datenbanken) oder auf Tabellenebene definiert werden. Wenn Sie die Verschlüsselung auf Datenbankebene definieren, müssen Sie die Verschlüsselungmethode und das Kennwort nur einmal festlegen, wenn Sie die Datenbank erstellen. Der Benutzer muss das Kennwort nur einmal für alle Tabellen der Datenbank eingeben. Folglich ist dies die empfohlene Vorgehensweise.

Frühere Versionen von TurboDB verlangten beides, ein Passwort und eine Code genannte 32-bit Zahl, um eine verschlüsselte Tabelle zu öffnen. Aktuelle Versionen erfordern nur eine Zeichenkette, das Passwort. Um kompatibel zu sein wird die frühere Kombination aus Passwort und Code zu einer Zeichenkette in diesem Format verknüpft: <Passwort>;<Code>. So werden das Passwort secret und der Code -3871 jetzt als das Password secret;-3871 eingegeben.

Hier eine Liste der verfügbaren Sicherheitseinstellungen. Diese werden in den Komponenten-Bibliotheken durch den Aufzählungswert für den Verschlüsselungstyp (encryption method) angegeben.

Name	Beschreibung	Schlüssel
Default	Für Tabellen in verwalteten Datenbanken: Bezieht die Einstellungen zur	Siehe jeweilige Zeile

	Verschlüsselung von der Datenbank. Andere Tabellen und Datenbanken: Keine Verschlüsselung	
None	Weder Verschlüsselung noch Passwortschutz	-
Protection	Die Tabelle ist nicht verschlüsselt aber mit Passwort geschützt.	Das Passwort, z.B. 3Huv
Classic	Die Tabelle ist mit einem 32 Bit Schlüssel verschlüsselt und mit Passwort geschützt.	Das Passwort und der nummerische Schlüssel durch Strichpunkt getrennt, z.B. 3Huv;97809878
Fast	Die Tabelle ist mit einer sehr schnellen 32-Bit Chiffre verschlüsselt. Ausreichend für vielen Anwendungen aber nicht 100% sicher.	Ein alphanummerisches Passwort bis zu 40 Zeichen, z.B. 3Huv
Blowfish	Verschlüsselung mit dem bekannten Blowfish Algorithmus, wobei ein 128 Bit Schlüssel verwendet wird.	Ein alphanummerisches Passwort bis zu 40 Zeichen, z.B. 3Huv
Rijndael	Verschlüsselung mit dem bekannten Rijndael Algorithmus, wobei ein 128 Bit Schlüssel verwendet wird. Bekannt unter Advanced Encryption Standard (AES).	Ein alphanummerisches Passwort bis zu 40 Zeichen, z.B. 3Huv
AES	Wie Rijndael.	Wie Rijndael

5.2.9.3 Sprachunabhängigkeit

TurboDB verfügt über eine Schnittstelle für Sprachtreiber, die eine sprachspezifische Vergleichssequenz für Sortierung, Suche und Indizierung implementieren können. Sprachtreiber sind Dynamische Link Bibliotheken. Die Sprachtreiber-Schnittstelle ist im TurboDB Sprachtreiber Toolkit beschrieben, das auf unserer Website zur Verfügung steht.

Eine Tabelle wird bei der Erzeugung oder Restrukturierung an einen Sprachtreiber gebunden. Daher bieten alle Werkzeuge zum Erzeugen von Tabellen eine Sprachauswahl als Eigenschaft der Tabelle.

Anwender der VCL Edition von TurboDB können auch die Texte der Fehlermeldungen an Ihre Bedürfnisse anpassen. Die Delphi-Quelltextdatei, in der die Texte untergebracht sind (*TdbMessages.pas*) wird mit ausgeliefert und kann für weitere Sprachen erweitert werden, indem den vordefinierten Konstanten entsprechende Texte zugewiesen werden.

Kompatibilität

Nicht unterstützt in TurboDB Managed.

5.3 TurboPL Guide

TurboDB Engine verfügt über einen Satz nativer Funktionen. Diese Funktionen wurden bis Tabellen-Level 4 zur Formulierung von Gültigkeitsbedingungen, berechneten Felder und Indexen und Vorgabewerten verwendet. Einige davon können aus Gründen der Kompatibilität auch in TurboSQL verwendet werden, das wird aber nicht empfohlen.

- Operatoren und Funktionen
- Suchbedingungen

5.3.1 Operatoren und Funktionen

- Arithmetische Operatoren und Funktionen
- String Operatoren und Funktionen
- Datum und Zeit Operatoren und Funktionen
- Sonstige Operatoren und Funktionen

5.3.1.1 TurboPL Arithmetische Operatoren und Funktionen

Die folgenden arithmetischen Operatoren und Funktionen können in TurboPL Ausdrücken eingesetzt werden. Sie werden nicht mehr für TurboSQL empfohlen

Operatoren

+	Addition
-	Subtraktion
*	Multiplikation
/	Fließkomma Division
div	Ganzzahl Division
mod	Modulo

Vergleiche

<	kleiner
<=	kleiner oder gleich
=	gleich
>=	größer oder gleich
>	größer
less	kleiner
is	gleich
greater	größer
<>	ungleich
from...upto	testet auf einen Bereich
in [...]	testet, ob ein Element in einer Menge enthalten ist

Functions

Abs(X: Real): Real	Liefert den absoluten Wert des Arguments, X.
ArcTan(X: Real): Real	Liefert den ArcTan der gegebenen Zahl X.
Cos(X: Real): Real	Liefert den Cosinus des Winkels X, im Bogenmaß.
Exp(X: Real): Real	Liefert die Exponentialfunktion der übergebenen Zahl.
Frac(X: Real): Real	Ermittelt den gebrochenzahligen Anteil der Zahl.
Int(X: Real): Integer	Liefert den ganzzahligen Anteil einer Zahl.
Log(X: Real): Real	Berechnet den natürlichen Logarithmus der Zahl.
Round(X: Real; [Precision: Integer]): Real	Rundet die angegebenen Zahl auf eine bestimmt Zahl an Nachkommastellen.
Sin(X: Real): Real	Berechnet den Sinus einer Zahl.

Sqrt(X: Real): Real	Berechnet die Quadratwurzel einer Zahl.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

5.3.1.2 TurboPL String Operatoren und Funktionen

Diese String Operatoren und Funktionen können in TurboPL Ausdrücken eingesetzt werden. Sie werden nicht mehr für TurboSQL empfohlen

Operatoren

+	String-Addition, z.B. EMPLOYEES.FirstName + ' ' + EMPLOYEES.LastName
[]	Zugriff auf einzelne Zeiche, z.B. EMPLOYEES.FirstName[1] + '. ' + EMPLOYEES.LastName

Vergleiche

<	kleiner
<=	kleiner oder gleich
=	gleich
>=	größer oder gleich
>	größer
less	kleiner
equal	gleich
greater	größer
<>	ungleich
has	sucht einen String innerhalb eines anderen mit Berücksichtigung von Groß- und Kleinschreibung, z.B: 'John Smith' has 'Sm'
like	vergleicht ohne Berücksichtigung von Groß/Klein-Schreibung mit einer Maske, die auch Joker-Zeichen enthalten kann, z.B: 'Smith' like 'sMIth', 'Smith' like 'Sm*', 'Smith' like 'Smit?' (alle wahr)
in [..]	testet, ob ein Element in einer Menge enthalten ist

Funktionen

Arguments in brackets are optional.

Asc(C: String): Integer	ANSI-Code des Zeichens.
Chr(N: Integer): String	Zeichen mit übergebenen ANSI-Code.
Exchange(Source, From, To: String): String	In der *Source* werden alle Vorkommen von From durch *To* ersetzt
FillStr(Source, Filler: String; Len: Integer): String	Füllt die Zeichenkette *Source* mit *Filler* bis zur durch *Length* gegebenen Länge und gibt das Ergebnis zurück.
LeftStr(Source: String; Len: Integer): String	Gibt die linken *Len* Zeichen von *Source* zurück.
Length(Source: String)	Liefert die Anzahl der Zeichen in *Source*.
Lower(Source: String): String	Liefert die Zeichenkette in Kleinbuchstaben konvertiert.
LTrim(Source: String): String	Gibt *Source* ohne führende Leerzeichen zurück.
MemoStr(Memo: MemoField [; Len: Integer]): String	Liefert die ersten 255 Zeichen des Inhalts eines *Memofeldes* des aktuellen Datensatzes.
NTimes(Source: String; Count: Integer): String	Liefert einen String, in dem die mit *Source* gegebene Zeichenkette *Count* mal wiederholt wird.
RealVal(Str: String): Real	Berechnet den nummerischen Wert eines String-Ausdrucks.
Pos(SubStr, Source: String):	Liefert die Position der Zeichenkette *SubStr* in Source oder 0,

Integer	falls *SubStr* nicht in *Source* enthalten ist.
RightStr(Source: String; Len: Integer)	Liefert die *Len* letzten Zeichen von Source.
RTrim(Source: String): String	Liefert *Source* ohne abschließende Leerzeichen.
Scan(SubStr, Source: String): Real	Zählt, wie oft *SubStr* in *Source* vorkommt.
Str(Num: Real[; Width, Precision: Real]): String	Wandelt *Num* in eine Zeichenkette um. *Width* gibt die Länge der Zeichenkette an, innerhalb derer die Zahl rechtsbündig dargestellt wird. Reicht die Länge zur Darstellung der Zahl nicht aus, wird die Zeichenkette automatisch soweit nötig erweitert. *Precision* bestimmt die Anzahl der Nachkommastellen. Die alphanummerischen Werte eines Aufzählungsfeldes (dtEnum) (siehe Datentypen für Tabellenspalten) können abenfalls mit *Str* ermittelt werden.
Upper(Source: String): String	Liefert den übergebenen String in Großbuchstaben.
NewGuid: String	Liefert eine Zeichenkette, die einen neuen Globally Unique Identifier bezeichnet.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

5.3.1.3 TurboPL Datum- und Zeit-Operatoren und Funktionen

Diese Datum und Zeit Operatoren und Funktionen können in TurboPL Ausdrücken eingesetzt werden. Sie werden nicht mehr für TurboSQL empfohlen

Vergleiche

Alle nummerischen Operatoren können genauso auch für Datum- und Zeit-Werte verwendet werden. Z.B. Datum 1 > Datum2.

Berechnungen

Sie können Zeitspannen einem Datum hinzufügen oder von einem Datum subtrahieren und Datums-Werte voneinander subtrahieren, um die Zeitspanne zu erhalten. Eine Zeitspanne ist eine Fließkommazahl, die eine Anzahl an Tagen repräsentiert (einschließlich des Nachkommaanteils für die Tageszeit) beim Rechnen mit Datums- und DateTime-Werten oder die Anzahl der Minuten (einschließlich des Nachkommaanteils für Sekunden und Millisekunden), wenn mit Zeit-Werten gerechnet wird.

Falls Time1 und Time2 Zeit-Variablen, Date1 und Date2 Datum-Variablen, DateTime1 und DateTime2 Variablen vom Typ DateTime und TimeSpan1 und TimeSpan2 Real Variablen sind, sind die folgenden Ausdrücke sinnvoll:

```
Time2 - Time1
Time2 - TimeSpan1
Time1 + TimeSpan2
Date2 - Date1
Date2 - TimeSpan1
Date2 + TimeSpan2
DateTime2 - DateTime1
DateTime2 - TimeSpan1
DateTime2 - TimeSpan2
```

Über die nummerischen Operatoren und Funktionen hinaus gibt es auch noch spezielle Datum- und Zeit-Funktionen:

Funktion	Beschreibung
CombineDateTime(ADate: Date; ATime: Time): DateTime	Kombiniert ein Datum und eine Zeit zu einem DateTime-Wert
Day(ADate: DateTime): Integer	Extrahiert den Tag aus einem Datum.
Hour(ADate: DateTime): Integer	Extrahiert die Stunden aus einem Time- oder DateTime-Wert

Millisecond(ADate: DateTime): Integer	Extrahiert den Millisekunden-Anteil aus einem Time- oder DateTime-Wert
Minute(ADate: DateTime): Integer	Extrahiert die Minuten aus einem Time- oder DateTime-Wert
Month(Date: DateTime): Integer	Extrahiert den Monat aus dem Datum.
Now: Time	Ermittelt die aktuelle Uhrzeit.
Second(ADate: DateTime): Integer	Extrahiert die Sekunden aus einem Time- oder DateTime-Wert.
Today: Date	Liefert das aktuelle Datum.
Week(Date: DateTime): Integer	Ermittelt die Wochen-Nummer für das Datum.
WeekDayNo(ADateTime: DateTime): Integer	Liefert den Wochentag als Nummer zwischen 1 (Montag) und 7 (Sonntag)
Year(Date: DateTime): Integer	Extrahiert das Jahr aus einem Datum.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

5.3.1.4 TurboPL Sonstige Operatoren und Funktionen

Darüberhinaus gibt es weitere Operatoren und Funktionen, die in TurboPL Ausdrücken eingesetzt werden können. Sie werden nicht mehr für TurboSQL empfohlen

Funktion	Beschreibung
HexStr(Value: Integer [; Digits: Integer])	Liefert die hexadezimale Darstellung einer Zahl.
CurrentRecordId(TableName)	Gibt den letzten vergebenen Wert für die AutoInc-Nummer der Tabelle zurück. Mit dieser Funktion kann man in einem einzigen zusammengesetzten Statement verknüpfte Datensätze in mehrere Tabellen eintragen.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

5.3.2 Suchbedingungen

- Filter Suchbedingungen
- Volltext Suchbedingungen

5.3.2.1 Filter Suchbedingungen

Filter-Suchbedingungen sind boolsche Ausdrücke, wie sie auch in SQL oder Pascal zum Einsatz kommen. Die Syntax und die verfügbaren Optionen sind identisch mit den in TurboSQL möglichen Bedingungen in Where-Klauseln mit zwei Ausnahmen, die zur besseren Kompatibilität mit BDE-Filtern eingeführt wurden:

- Datum-, Zeit- und nummerische Formate basieren auf den lokalen Einstellungen des Systems
- * und ? sind als Joker ebenso erlaubt wie % und _.
- Mengen werden in eckigen statt in runden Klammern geschrieben

Beispiele (Name, Betrag, Betrag1, Betrag2 und Geburtstag sind Tabellenspalten)

```
Name = 'Schmidt'
Name like 'Schmi*'
Name like 'Schmi%'
Name like 'Schmid?'
Name like 'Schmid_'
Name has 'mid'
LeftStr(Name, 2) = 'Sc'
```

```
Length(Name) > 4
Betrag = 13546,45
Betrag < 13546,46
Betrag < 345,67 or Betrag > 567,89 (Das Dezimaltrennzeichen hängt von den
lokalen Einstellungen des Systems ab)
Betrag1 * 0,3 > Betrag2 * 0,8
Betrag is not null (alle Datensätze, die für Betrag keinen Wert haben)
Geburtstag = "20.4.1962" (Das Datum-Format hängt von den lokalen Einstellungen
des Systems ab.)
Geburtstag < "20.4.1962"
Geburtstag between "1.4.1962" and "30.4.1962"
Year(Geburtstag) = 1962
Date-of-birth is null  (alle Datensätze, die für Geburtstag keinen Wert haben)
```

Regeln für Anführungszeichen und Escaping

Sowohl einfache als auch doppelte Anführungszeichen können benutzt werden. Anführungszeichen innerhalb von Zeichenketten, die von denselben Anführungszeichen umschlossen sind, sind zu duplizieren:

```
"My ""quote""" -> My "quote"
'My "quote"' -> My "quote"
'My ''quote''' -> My 'quote'
"My 'quote'" -> My 'quote'
```

Falls der Name einer Tabellen-Spalte, auch ein TurboSQL Schlüsselwort ist, muss der Bezeichner entweder in doppelten Anführungszeichen oder mit einem führenden $ geschrieben werden:

```
Length($Password) > 8
```

TurboDB bietet leistungsstarke Funktionen und Operatoren zum Einsatz in Suchbedingungen, z.B. *like*, *from...upto*, *LeftStr*, *Year* und viele mehr. In Operatoren und Funktionen finden Sie eine komplette Referenz. Bedingungen können mit den logischen Operatoren a*nd*, *or* und *not* kombiniert werden.

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

5.3.2.2 Volltext Suchbedingungen

Volltext Suchbedingungen werden im TurboSQL CONTAINS Prädikat und in der VCL *TTdbTable.WordFilter* Property verwendet. Eine Volltext-Suchbedingung ist im Wesentlichen eine Liste von Schlüsselwörtern, voneinander getrennt durch "+", "," oder "-". Diese Zeichen stehen für:

, oder Leerzeichen	beide Schlüsselwörter müssen im Datensatz vorhanden sein (und Verknüpfung)
+ oder /	eines der Schlüsselwörter muss im Datensatz zu finden sein (oder Verknüpfung)
-	Das Schlüsselwort darf nicht im Datensatz vorhanden sein (nicht Operator)

Die alternativen Zeichen (Leerzeichen und /) sind nur ab Tabellen-Level 4 verfügbar. Schlüsselwörter dürfen die Joker "?" und "*" enthalten um entweder ein beliebiges Zeichen oder eine beliebige Zeichenfolge zu repräsentieren.

Beispiele

Database	Findet *Database*, *database*, *dataBase*, ...
Database*	Findet *database*, *Databases*, *DatabaseDriver*, ...
Data?ase	Findet *Database*, *dataCase*, ...
Database, Driver	Im Datensatz müssen die Wörter *Database* and *Driver* vorkommen
Database Driver	Wie oben, für Tabellen-Level 4
Database, Driver, ODBC	Im Datensatz müssen die Wörter *Database*, *Driver* and *ODBC* vorkommen
Database Driver ODBC	Wie oben, für Tabellen-Level 4
Database + Driver	Im Datensatz müssen das Wort *Database* oder das Wort *Driver* oder

	beides vorkommen
Database/Driver	Wie oben, für Tabellen-Level 4
Database + Driver + ODBC	Im Datensatz muss entweder *Database* oder *Driver* oder *ODBC* vorkommen
Database/Driver/ODBC	Wie oben, für Tabellen-Level 4
Database Driver ODBC/OLE	Im Datensatz müssen die Wörter *Database* und *Driver* und entweder *ODBC* oder *OLE* vorkommen
-Database	Im Datensatz darf das Wort *Database* nicht vorkommen
Database - Driver	Im Datensatz muss das Wort *Database* vorkommen, *Driver darf* dagegen nicht enthalten sein

Kompatibilität

TurboPL wird nur zur Rückwärtskompatibilität in Tabellen bis Level 4 unterstützt.

5.4 TurboSQL Guide

Turbo SQL ist eine Untermenge von SQL 92, die in der MS ODBC Spezifikation enthaltene SQL Definition und ist sehr ähnlich dem in der Borland Database Engine enthaltenem Local SQL.

Konventionen

- Tabellennamen
- Spaltennamen
- String Literale
- Datumsformate
- Zeitformate
- DateTime Format
- Boolsche Konstanten
- Tabellenkorrelationsnamen
- Spaltenkorrelationsnamen
- Parameter
- Eingebettete Kommentare

Data Manipulation Language

- Überblick
- DELETE Statement
- FROM Klausel
- GROUP BY Klausel
- INSERT Statement
- ORDER BY Klausel
- SELECT Statement
- UPDATE Statement
- WHERE Klausel

Data Definition Language

- Überblick
- CREATE TABLE Kommando
- ALTER TABLE Kommando
- CREATE INDEX Kommando
- DROP Kommando
- Datentypen für Tabellenspalten

Programmiersprache

- Überblick
- CALL Statement
- CREATE FUNCTION Statement
- CREATE PROCEDURE Statement
- CREATE AGGREGATE Statement
- DROP FUNCTION/PROCEDURE/AGGREGATE Statement
- DECLARE Statement
- IF Statement
- SET Statement
- WHILE Statement
- Parameteraustausch mit .NET Assemblies

5.4.1 TurboSQL vs. Local SQL

TurboSQL unterscheidet sich in einigen Punkten von Borland's Local SQL:

  - In TurboSQL können Sie Datums, Zeit und Datetime Literaleohne Anführungszeichen angeben, das Format ist tt.mm.jjjj und ss:mm und tt.mm.jjjj_hh:mm:ss.ms
  - Mit TurboDB SQL können mehrere durch Semikolon getrennte Kommandos in eiunem Statement zusammengefasst werden.
- TurboDB kann Spalten bestehender Tabellen mit Hilfe des ALTER TABLE Kommandos umbenennen und ändern werden.

5.4.2 Konventionen

5.4.2.1 Tabellennamen

TurboDB beschränkt wie der ANSI-Standard-SQL alle Tabellennamen auf ein einzelnes Wort, das sich aus alphanumerischen Zeichen und dem Unterstrich, "_", zusammensetzt.

```
SELECT *
FROM customer
```

TurboSQL unterstützt vollständige Datei- und Pfadangaben in Tabellenreferenzen. Tabellenreferenzen über Pfad oder Dateiname und Erweiterung müssen in einfachen oder doppelten Anführungszeichen eingeschlossen werden. Zum Beispiel:

```
SELECT *
FROM 'parts.dat'

SELECT *
FROM "c:\sample\parts.dat"
```

Falls Sie die Dateierweiterung für den Namen einer Tabelle weglassen, wird automatisch ".dat" angehängt.

5.4.2.2 Spaltennamen

Wie der ANSI-Standard-SQL beschränkt TurboDB alle Spaltennamen auf ein einzelnes, aus alphanumerischen Zeichen und dem Unterstrich, "_", bestehendes Wort. Um gleiche Spaltennamen in verschiedenen Tabellen zu unterscheiden können Sie den Tabellennamen vor dem Spaltenbezeichner angeben.

```
SELECT Employee_Id
FROM Employee
```

oder

```
SELECT Employee.Employee_Id
FROM Employee
```

Außerdem kann TurboSQL auch deutsche Umlaute für Tabellen- und Spaltennamen benutzen:

```
SELECT Kürzung
FROM Beiträge
```

Für Spaltennamen, die Leerzeichen oder Spezialzeichen enthalten und um zwischen Spaltennamen und Schlüsselwörtern unterscheiden zu können, kann der Spaltenname in doppelten Anführungszeichen oder eckigen Klammern geschrieben werden.

```
SELECT "Employee Id", [Employee Id]
FROM [Update]
```

5.4.2.3 String Literale

Zeichenketten Literale sind von einfachen Anführungszeichen umschlossen. Um ein einzelnes Anführungszeichen innerhalb einer Zeichenkette zu bezeichnen, sind zwei einzufügen. Hier Beispiele für gültige Zeichenketten Literale:

```
'This is a string literal'
'This is a ''single-quoted'' string literal'
'This is a "double-quoted" string literal'
```

Ein ungültiges Beispiel:

```
'This isn't a valid string literal'
```

Hinweis

In früheren TurboDB Versionen waren auch doppelte Anführungszeichen für Zeichenketten Literale möglich. Für eine bessere SQL-Kompatibilität werden Doppelt-Anführungsstriche jetzt nur noch für Tabellen- und Spaltenbezeichner verwendet.

5.4.2.4 Datumsformate

TurboDB kennt zwei unterschiedliche Notationen für Datumskonstanten. Das native Format ist tt.mm.jjjj. Dieses Format entspricht der natürlichen Schreibweise und kann vom Parser nicht mit einem arithmetischen Ausdruck verwechselt werden.

Aus diesem Grund, ist es unnötig (sogar verboten) solch ein Datumsliteral in Anführungszeichen zu setzen. Beispiel:

```
SELECT *
FROM orders
WHERE saledate <= 31.12.2001
```

sucht nach Verkäufen bis zum 31 Dezember 2001

Falls Sie das Datums im amerikanischem Format angeben wollen, z.B. 12/31/2001 oder 2001-12-31, müssen Sie es mit einfachen Anführungszeichen umschließen und das Schlüsselwort DATE voranstellen:

```
SELECT *
FROM orders
WHERE saledate <= DATE'12/31/2001'
```

oder

```
SELECT *
FROM orders
WHERE saledate <= DATE'2001-12-31'
```

Das deutsche Format funktioniert genauso:

```
SELECT * FROM orders
WHERE saledate <= DATE'31.12.2001'
```

Führende Nullen bei Monats- und Tagesfeldern sind optional. Wird beim Jahr nicht das Jahrhundert angegeben, wird das 20te Jahrhundert für die Jahre von 50 bis 99 und das 21te Jahrhundert für die Jahre von 00 bis 49 angenommen.

Das Schlüsselwort *DATE* kann entfallen, wenn der Typ der Zeichenkette wie in den beiden obigen Beispielen offensichtlich ist.

Beispiel

```
SELECT *
FROM orders
WHERE (saledate > 1.1.89) AND (saledate <= 31.12.20)
```

sucht nach Verkäufen zwischen dem 1. Januar 1989 und dem 31. Dezember 2020.

5.4.2.5 Zeitformate

Turbo SQL bietet zwei verschiedene Notationen zur Angabe von Zeitkonstanten. Das native Format erwartet Zeitliterale in der Form hh:mm wobei hh für die Stunden und mm für die Minuten steht. Turbo SQL benutzt die 24 Stunden Skala, das bedeutet 2:10 ist früh am morgen (2:10 AM) während 14:10 am frühen Nachmittag ist (2:10 PM).Der Zeitausdruck darf nicht in Anführungszeichen stehen.

```
INSERT INTO WorkOrder
(ID, StartTime)
VALUES ('B00120', 22:30)
```

Falls Sie es vorziehen, können Sie die Zeit auch im amerikanischen *hh.mm am/pm* Format angeben.Dann müssen Sie das Literal mit Anführungszeichen umgeben und das Schlüsselwort *TIME* voranstellen:

```
INSERT INTO WorkOrder
(ID, StartTime)
VALUES ('B00120', TIME'10.30 pm')
```

Ist der Typ der Zeichenkette offensichtlich, wie im vorangegangenen Beispiel, kann das Schlüsselwort *TIME* weggelassen werden. Im folgenden Beispiel dagegen muss *TIME* stehen:

```
SELECT StartTime - TIME'12:00:00 pm' FROM WorkOrder
```

Hinweis

Die erste Alternative, das native Format ohne Anführungszeichen, kann nicht mit der VCL-Komponente *TTdbQuery* verwendet werden. Der VCL-Parser für SQL interpretiert den Doppelpunkt als Startzeichen eines Parameters und meldet einen Fehler.

5.4.2.6 DateTime Format

Das native Format für DateTime Literale setzt sich aus einem Datums- und Zeitanteil zusammen, getrennt duch einen Unterstrich. Anführungszeichen können optional gesetzt werden.

```
SELECT *
FROM WorkOrder
WHERE StartTime >= 31.1.2001_14:10:00
```

oder

```
SELECT *
FROM WorkOrder
WHERE StartTime >= '31.1.2001_14:10:00'
```

Eine andere Möglichkeit ist es ein DateTime in einfache Anführungszeichen zu setzen und das Schlüsselwort TIMESTAMP voranzustellen. Auch hier gibt es drei verschiedene Möglichkeiten der Darstellung:

Das amerikanische Format

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'1/31/2001 2:10:00 pm'
```

Das internationale Format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'2001-1-31 14:10:00'
```

Das deutsche Format:

```
SELECT * FROM WorkOrder
WHERE StartTime >= TIMESTAMP'31.1.2001 14:10:00'
```

Wenn, wie in den obigen Beispielen, die Natur der DateTime-Zeichenkette eindeutig ist, kann das Schlüsselwort TIMESTAMP weggelassen werden.

Hinweis

Die erste Variante mit dem nativen Format ohne umschließende Klammer kann mit der Delphi-Komponente TTdbQuery nicht benutzt werden. Der VCL-Parser für SQL-Befehle interpretiert den Doppelpunkt als Startzeichen eines Parameters und erzeugt eine Fehlermeldung.

5.4.2.7 Boolsche Konstanten

Die Boolschen Konstantenwerte TRUE und FALSE müssen ohne Anführungszeichen angegeben werden. Eine Unterscheidung zwischen Groß- und Kleinschreibung wird nicht vorgenommen.

```
SELECT *
FROM transfers
WHERE (paid = 'True') AND NOT (incomplete = FALSE)
```

5.4.2.8 Tabellenkorrelationsnamen

Tabellenkorrelationsnamen werden verwendet, um eine Spalte explizit mit der Tabelle zu verknüpfen, aus der sie stammt. Dies ist besonders nützlich, wenn mehrere Spalten gleichen Namens in derselben Anfrage erscheinen, üblicherweise in Mehrfach-Tabellenabfragen. Ein Tabellenkorrelationsname wird definiert, indem der Tabellenreferenz in FROM-Klausel einer SELECT-Abfrage ein eindeutiger Bezeichner nachgestellt wird. Dieser Bezeichner oder Tabellenkorrelationsname kann dann verwendet werden, um einem Spaltennamen vorangestellt zu werden.

Ist der Tabellenname kein String in Anführungszeichen, so ist der Tabellenname der implizit vorgegebene Korrelationsname. Ein expliziter Korrelationsname, welcher mit dem Tabellennamen übereinstimmt, muß in der FROM-Klausel nicht angegeben werden, und der Tabellenname kann Spaltennamen in anderen Teilen der Anweisung vorangestellt werden.

```
SELECT *
FROM "/home/data/transfers.dat" transfers
WHERE transfers.incomplete = False
```

5.4.2.9 Spaltenkorrelationsnamen

Verwenden Sie das Schlüsselwort AS, um einer Spalte, einem aggregierten Wert oder einer Konstante einen Korrelationsnamen zuzuweisen. In der folgenden Anweisung sind die Token Sub und Word Spaltenkorrelationsnamen.

```
SELECT SUBSTRING(company FROM 1 FOR 1) AS sub, Text word
FROM customer
```

5.4.2.10 Parameter

Befehls-Parameter werden durch einen Doppelpunkt eingeleitet:

```
INSERT INTO Customer (Name) VALUES(:Name)
```

Der Name des Parameters ist der Bezeichner ohne den Doppelpunkt, d.h. Name im obigen Fall. Um auf den Parameter in einer API-Funktion oder von einer Komponenten-Bibliothek aus zuzugreifen, wird der Bezeichner ohne Doppelpunkt benutzt.

Bei der Arbeit mit dem ODBC Interface, werden auch unbenannte Parameter unterstützt:

```
INSERT INTO Customers (Name) VALUES (?)
```

5.4.2.11 Eingebettete Kommentare

Es gibt zwei Möglichkeiten Text als Kommentar in ein SQL Statement einzubetten, die vergelichbar mit den Konventionen unter C++ sind. Entweder Sie benutzen /* und */ um auch mehrzeiligen Kommentar zu umschließen oder Sie beginn den Kommentar mit //, das Ende wird durch das Ende der Zeile bestimmt.

Beispiel:

```
/* This finds all requests that have come in in the period of time we are
looking at. */
SELECT * FROM Request
// Requests from the time before the Euro came
WHERE Date < '1/1/2002'
```

5.4.3 Data Manipulation Language

Turbo SQL unterstützt folgende Data Manipulation Language- (DML-) Anweisungen, Klauseln, Funktionen und Prädikate:

Statements

DELETE	Löscht bestehende Datensätze aus einer Tabelle.
INSERT	Fügt einer Tabelle neue Daten hinzu
SELECT	Liest bestehende Daten aus einer Tabelle aus.
UPDATE	Verändert bestehende Daten in einer Tabelle..

Klauseln

FROM	Gibt die für die Anweisung verwendeten Tabellen an.
GROUP BY	Gibt die Spalten an, die zum Gruppieren von Zeilen verwendet werden.
HAVING	Gibt Filterkriterien unter Verwendung aggregierter Daten an.
ORDER BY	Gibt die Spalten an, anhand deren die Ergebnismenge zu sortieren ist.
WHERE	Gibt Filterkriterien zum Begrenzen der abgerufenen Zeilen an.
Unterabfragen	Vergleich mit dem Ergebnis einer Abfrage mit IN, ANY, SOME, ALL und EXISTS.

Funktionen, Prädikate und Operatoren

Allgemeine Funktionen und Operatoren	Berechnungen und Vergleiche mit Zahlen, Zeichenketten, Datum- und Zeitwerten etc.
Arithmetische Funktionen und Operatoren	Berechnungen mit Zahlen
Zeichenketten Funktionen und Operatoren	Berechnungen mit Zeichenketten
Datum und Zeit Funktionen und Operatoren	Berechnungen mit Datum- und Zeitwerten
Aggregat Funktionen	Statistik
Sonstige Funktionen und Operatoren	Berechnungen, die sich nicht in eine der Kategorien einordnen lassen
Tabellen Operatoren	Kombiniert zwei Tabellen zu einer neuen
Unterabfragen	Vergleicht Datensätze mit der Ergebnismenge einer anderen Abfrage
Volltextsuche	Ausführen einer Volltextsuche

5.4.3.1 DELETE Anweisung

Löscht eine oder mehrere Zeilen aus einer Tabelle.

```
DELETE FROM table_reference
[WHERE predicates]
```

Beschreibung

Verwenden Sie DELETE, um eine oder mehrere Zeilen aus einer Tabelle zu löschen.

```
DELETE FROM "employee.dat"
```

Die optionale WHERE-Klausel beschränkt das Löschen von Zeilen auf einen Teil der Zeilen in der Tabelle. Ist keine WHERE-Klausel angegeben, so werden alle Zeilen in der Tabelle gelöscht:

```
DELETE FROM "employee.dat"
WHERE empno > 2300
```

Die Tabellenreferenz kann der DELETE-Anweisung nicht als Parameter übergeben werden.

Wichtiger Hinweis:

Das DELETE-Statement ohne WHERE-Klausel löscht alle Zeilen eine Tabelle ohne Einschränkungen wie z.B. Fremdschlüssel zu prüfen. Dies ist ein Leistungsmerkmal zum schnellen Leeren von Tabellen.

5.4.3.2 FROM Klausel

Gibt die Tabellen an, aus denen eine SELECT-Anweisung Daten entnimmt.

```
FROM table_reference [, table_reference...]
```

Beschreibung

Verwenden Sie eine FROM-Klausel, um die Tabellen anzugeben, aus denen eine SELECT-Anweisung Daten entnimmt. Der Wert für eine FROM-Klausel ist eine durch Kommas getrennte Liste von Tabellennamen. Angegebene Tabellennamen müssen den Benennungskonventionen von Turbo SQL entsprechen. Beispielsweise entnimmt die folgende SELECT-Anweisung Daten aus einer einzelnen TurboDB-Tabelle:

```
SELECT *
FROM "customer.dat"
```

```
SELECT * FROM
customer, orders
```

```
SELECT * FROM
customer JOIN orders ON orders.CustNo = customer.CustNo
```

Anwendung

SELECT

5.4.3.3 GROUP BY Klausel

Verbindet Zeilen mit gemeinsamen Spaltenwerten in einzelnen Zeilen.

```
GROUP BY column_reference [, column reference...]
```

Beschreibung

Verwenden Sie eine GROUP BY-Klausel zum Verbinden von Zeilen mit denselben Spaltenwerten in einer einzigen Zeile. Die Kriterien zum Verbinden von Zeilen basieren auf den Werten in denjenigen Spalten, die in der GROUP BY-Klausel angegeben sind. Der Zweck der Verwendung einer GROUP BY-Klausel besteht darin, einen oder mehrere Spaltenwerte in einem einzelnen Wert zu verbinden (aggregieren) und eine oder mehrere Spalten zum eindeutigen Identifizieren der aggregierten Werte bereitzustellen. Eine GROUP BY-Klausel kann nur verwendet werden, wenn auf eine oder mehrere Funktionen eine Aggregatfunktion angewendet wird.

Der Wert für die GROUP BY-Klausel ist eine durch Kommas getrennte Spaltenliste. Jede Spalte in dieser Liste muß folgenden Kriterien entsprechen:

- Sie muß in einer der in der FROM-Klausel der Abfrage angegebenen Tabellen stehen.
- Sie muß in der SELECT-Klausel der Abfrage stehen.
- Auf sie darf keine Aggregatfunktion angewendet werden.

Wird eine GROUP BY-Klausel verwendet, so müssen alle Tabellenspalten in der SELECT-Klausel der Abfrage mindestens einem der folgenden Kriterien genügen, oder sie darf nicht in der SELECT-Klausel stehen:

- Sie muß in der GROUP BY-Klausel der Abfrage stehen.
- Sie muß im Subjekt einer Aggregatfunktion stehen.

Für Literale in der SELECT-Klausel gelten die genannten Kriterien nicht.

Die Unterscheidbarkeit von Zeilen basiert auf den Spalten in der angegebenen Spaltenliste. Alle Zeilen mit gleichen Werten in diesen Spalten werden in einer einzigen Zeile (oder logischen Gruppe) kombiniert. Für Spalten, die Subjekt einer Aggregatfunktion sind, werden die Werte über alle Zeilen der Gruppe kombiniert. Alle Spalten, die nicht Subjekt einer Aggregatfunktion sind, behalten ihre Werte bei und dienen dazu, die Gruppe eindeutig zu identifizieren. So werden beispielsweise in der folgenden SELECT-Anweisung die Werte in der Spalte SALES basierend auf unterschiedlichen Werten in der Spalte COMPANY in Gruppen aggregiert (summiert). Auf diese

Weise werden Verkaufssummen für jede Firma berechnet:

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY company
ORDER BY company
```

Eine Spalte kann in einer GROUP BY-Klausel durch einen Spaltenkorrelationsnamen anstelle tatsächlicher Spaltennamen referenziert werden. Die nachfolgende Anweisung gruppiert unter Verwendung der ersten Spalte, COMPANY, repräsentiert durch den Spaltenkorrelationsnamen Co:

```
SELECT company AS Co, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY Co
ORDER BY 1
```

Hinweis

- Abgeleitete Werte (berechnete Felder) können nicht als Basis für eine GROUP BY-Klausel verwendet werden.
- Spaltenreferenzen können einer GROUP BY-Klausel nicht als Parameter übergeben werden.

Anwendung

SELECT, wenn Aggregatfunktionen verwendet werden

5.4.3.4 HAVING Klausel

Gibt Filterbedingungen für eine SELECT-Anweisung an.

```
HAVING predicates
```

Beschreibung

Verwenden Sie eine HAVING-Klausel, um die von einer SELECT-Anweisung abgerufenen Zeilen auf eine Untermenge von Zeilen zu beschränken, für welche die aggregierten Spaltenwerte den angegebenen Kriterien genügen. Eine HAVING-Klausel kann nur dann in einer SELECT-Anweisung verwendet werden, wenn:

- die Anweisung auch eine GROUP BY-Klausel enthält,
- eine oder mehrere Spalten Subjekte von Aggregatfunktionen sind.

Der Wert für eine HAVING-Klausel sind ein oder mehrere logische Ausdrücke oder Prädikate, die für jede aggregierte, der Tabelle entnommene Zeile TRUE oder FALSE ergeben. Nur jene Zeilen, für welche die Prädikate TRUE ergeben, werden von einer SELECT-Anweisung entnommen. So entnimmt beispielsweise folgende SELECT-Anweisung alle Zeilen, in denen die Verkaufssumme für individuelle Verkaufssummen den Wert 1000 übersteigt:

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
GROUP BY company
HAVING (SUM(sales) >= 1000)
ORDER BY company
```

Mehrere Prädikate müssen durch einen der logischen Operatoren OR oder AND getrennt werden. Jedes Prädikat kann durch den Operator NOT negiert werden. Klammern können zur Erzeugung verschiedener Zeilenevaluierungskriterien verwendet werden, um logische Vergleiche und Gruppen von Vergleichen zu isolieren.

Eine SELECT-Anweisung kann sowohl eine WHERE- als auch eine HAVING-Klausel enthalten. Die WHERE-Klausel filtert die zu aggregierenden Daten unter Verwendung der Spalten, die nicht Subjekte von Aggregatfunktionen sind. Die HAVING-Klausel filtert dann die Daten nach der Aggregation unter Verwendung derjenigen Spalten, die Subjekte von Aggregatfunktionen sind. Folgende SELECT-Abfrage führt die gleiche Operation wie obige durch, die Daten sind jedoch auf jene beschränkt, deren Spalte STATE den Wert "CA" hat:

```
SELECT company, SUM(sales) AS TOTALSALES
FROM sales1998
WHERE (state = "CA")
GROUP BY company
HAVING (SUM(sales) >= 1000)
ORDER BY company
```

Hinweis

Eine HAVING-Klausel filtert Daten nach der Aggregation einer GROUP BY-Klausel. Verwenden Sie eine WHERE-Klausel zum Filtern basierend auf Zeilenwerten vor der Aggregation.

Anwendung

SELECT mit GROUP BY

5.4.3.5 INSERT Anweisung

Fügt einer Tabelle eine oder mehrere Datenzeilen hinzu.

```
INSERT INTO table_reference
[(columns_list)]
VALUES (update_atoms)
```

Beschreibung

Verwenden Sie die Anweisung INSERT, um einer Tabelle eine oder mehrere Datenzeilen hinzuzufügen.

Verwenden Sie eine Tabellenreferenz in der INTO-Klausel, um die Tabelle anzugeben, welche die eingehenden Daten aufnehmen soll.

Bei der Spaltenliste handelt es sich um eine durch Kommas getrennte, in Klammern eingeschlossene, optionale Liste der Spalten in der Tabelle. Die VALUES-Klausel ist eine durch Kommas getrennte, in Klammern eingeschlossenen Liste von Aktualisierungsatomen. Ist keine Spaltenliste angegeben, so werden eingehende Aktualisierungswerte (Aktualisierungsatome) in den Feldern gespeichert, wie sie sequentiell in der Tabellenstruktur definiert sind. Aktualisierungsatome werden auf die Spalten in der Reihenfolge angewendet, in der die Aktualisierungsatome in der VALUES-Klausel aufgeführt sind. Außerdem müssen so viele Aktualisierungsatome vorhanden sein wie Spalten in der Tabelle.

```
INSERT INTO "holdings.dat"
VALUES (4094095, "BORL", 5000, 10.500, 2.1.1998)
```

Ist eine explizite Spaltenliste angegeben, so werden eingehende Aktualisierungsatome (in der Reihenfolge, in der sie in der VALUES-Klausel auftreten) in den aufgeführten Spalten (in der Reihenfolge, in der sie in der Spaltenliste auftreten) gespeichert. In etwaigen Spalten, die nicht in der Spaltenliste stehen, werden NULL-Werte gespeichert:

```
INSERT INTO "customer.dat"
(custno, company)
VALUES (9842, "dataWeb GmbH")
```

Um Tabellen Zeilen aus einer anderen Tabelle hinzuzufügen, lassen Sie das Schlüsselwort VALUES weg und verwenden Sie eine Unterabfrage als Quelle der neuen Zeilen:

```
INSERT INTO "customer.dat"
(custno, company)
SELECT custno, company
FROM "oldcustomer.dat"
```

5.4.3.6 ORDER BY Klausel

Sortiert die von einer SELECT-Anweisung abgerufenen Werte.

```
ORDER BY column_reference [, column_reference...] [ASC|DESC]
```

Beschreibung

Verwenden Sie die Klausel ORDER BY, um die von einer SELECT-Anweisung abgerufenen Zeilen basierend auf den Werten einer oder mehrerer Spalten zu sortieren.

Bei dem Wert für die Klausel ORDER BY handelt es sich um eine durch Kommas getrennte Liste von Spaltennamen. Die Spalten in dieser Liste müssen auch in der SELECT-Klausel der Abfrageanweisung stehen. Die Spalten in der ORDER BY-Liste können aus einer oder mehreren Tabellen kommen. Eine Zahl, welche die relative Position einer Spalte in der SELECT-Klausel repräsentiert, kann anstelle eines Spaltennamens verwendet werden. Spaltenkorrelationsnamen können ebenfalls in der Spaltenliste einer ORDER BY-Klausel verwendet werden.

Verwenden Sie ASC (bzw. ASCENDING), um festzulegen, daß die Sortierung in aufsteigender Reihenfolge erfolgen soll (vom Kleinsten zum Größten Wert). Verwenden Sie DESC (bzw. DESCENDING) für eine absteigende Sortierungsreihenfolge (vom Größten zum Kleinsten Wert).

Wenn nicht angegeben, wird ASC als Vorgabe verwendet.

Die folgende Anweisung sortiert die Ergebnismenge aufsteigend nach dem der Spalte LASTINVOICEDATE entnommenen Jahr, dann absteigend nach der Spalte STATE und dann aufsteigend nach der Spalte COMPANY, umgewandelt in Großbuchstaben:

```
SELECT EXTRACT(YEAR FROM lastinvoicedate) AS YY, state, UPPER(company)
FROM customer
ORDER BY YY DESC, state ASC, 3
```

Spaltenreferenzen können einer ORDER BY-Klausel nicht als Parameter übergeben werden.

Anwendung

SELECT

5.4.3.7 SELECT Anweisung

Liest Daten aus Tabellen aus.

```
SELECT [TOP number] [DISTINCT] * | column_list
FROM table_reference
[WHERE predicates]
[ORDER BY order_list]
[GROUP BY group_list]
[HAVING having_condition]
```

Beschreibung

Verwenden Sie die Anweisung SELECT, um

- aus einer Tabelle eine einzelne Zeile oder einen Teil einer Zeile abzurufen. Dies wird als Singleton-Select bezeichnet.
- aus einer Tabelle mehrere Zeilen oder Teile von Zeilen abzurufen.
- aus der Verbindung zweier oder mehrerer Tabellen verwandte Zeilen oder Teile von Zeilen abzurufen.

Die SELECT-Klausel definiert die Liste von Elementen, die von der SELECT-Anweisung zurückgegeben wird. Die SELECT-Klausel verwendet eine durch Kommas getrennte Liste, die sich aus folgenden Bestandteilen zusammensetzt: Spalten von Tabellen, Konstanten und durch Funktionen veränderte Spalten- oder Konstantenwerte. Konstantenwerte in der Spaltenliste können der SELECT-Anweisung als Parameter übergeben werden. Sie können Parameter nicht verwenden, um Spaltennamen zu repräsentieren. Verwenden Sie das Sternzeichen ("*"), um Werte aus allen Spalten auszulesen.

Die Spalten in der Spaltenliste für die SELECT-Klausel können aus einer oder mehreren Tabellen stammen, sofern diese Tabellen in der FROM-Klausel aufgeführt sind. Die FROM-Klausel identifiziert die Tabelle(n), aus der/denen die Daten ausgelesen werden.

Falls das Schlüsselwort TOP angegeben ist, wird die Anzahl der Datensätze der Ergebnismenge auf die gegebenen Zahl eingeschränkt. Top wird nach allen anderen Klauseln ausgewertet und beachtet daher die gewählte Sortierung oder Gruppierung, falls ORDER BY und/oder GROUP BY angegeben wurden.

Wenn das Schlüsselwort DISTINCT angegeben ist, werden doppelte Zeilen in der Ergebnis-Tabelle unterdrückt. DISTINCT kann nicht zusammen mit GROUP BY verwendet werden. Falls eine SELECT-Anweisung sowohl GROUP BY als auch DISTINCT enthält, wird das Schlüsselwort DISTINCT ignoriert.

Die folgende Anweisung entnimmt Daten für zwei Spalten aus allen Zeilen einer Tabelle:

```
SELECT custno, company
FROM orders
```

Siehe auch

JOIN, UNION, INTERSECT, EXCEPT

5.4.3.8 UPDATE Anweisung

Ändert eine oder mehrere bestehende Zeilen in einer Tabelle.

```
UPDATE table_reference
SET column_ref = update_atom [, column_ref = update_atom...]
[WHERE predicates]
```

Beschreibung

Verwenden Sie die Anweisung UPDATE, um einen oder mehrere Spaltenwerte in einer oder mehreren bestehenden Zeilen einer Tabelle zu verändern.

Verwenden Sie eine Tabellenreferenz in der UPDATE-Klausel, um die Tabelle anzugeben, welche die Datenänderungen erhalten soll.

Die SET-Klausel ist eine durch Kommas getrennte Liste von Aktualisierungsausdrücken. Jeder Ausdruck setzt sich aus dem Namen einer Spalte, dem Zuweisungsoperator (=) und dem Aktualisierungswert (Aktualisierungsatom) für diese Spalte zusammen. Die Aktualisierungsatome in einem jeden Aktualisierungsausdruck können Konstanten, Singleton-Rückgabewerte aus einer Unterabfrage oder durch Funktionen veränderte Aktualisierungsatome sein. Unterabfragen, die ein Aktualisierungsatom für einen Aktualisierungsausdruck liefern, müssen eine Singleton-Ergebnismenge (eine Zeile) zurückgeben und dürfen nur eine einzelne Spalte zurückgeben.

```
UPDATE salesinfo
SET taxrate = 0.0825
WHERE (state = 'CA')
```

Die optionale WHERE-Klausel beschränkt Aktualisierungen auf einen Teil der Zeilen in der Tabelle. Ist keine WHERE-Klausel angegeben, so werden alle Zeilen in der Tabelle unter Verwendung der Aktualisierungsausdrücke der SET-Klausel aktualisiert.

Siehe auch

INSERT, DELETE

5.4.3.9 WHERE Klausel

Gibt Filterbedingungen für eine SELECT- oder UPDATE-Anweisung an.

```
WHERE predicates
```

Beschreibung

Verwenden Sie eine WHERE-Klausel, um die Auswirkungen einer SELECT- oder UPDATE-Anweisung auf eine Untermenge von Zeilen in der Tabelle zu beschränken. Die Verwendung einer WHERE-Klausel ist optional.

Der Wert für eine WHERE-Klausel ist ein oder mehrere logische Ausdrücke oder Prädikate, die für jede Zeile in der Tabelle TRUE oder FALSE ergeben. Nur jene Zeilen, für welche die Prädikate TRUE ergeben, werden von einer SELECT-Anweisung entnommen oder von einer UPDATE-Anweisung aktualisiert. Beispielsweise entnimmt die folgende SELECT-Anweisung alle Zeilen, für welche die Spalte STATE den Wert "CA" enthält:

```
SELECT company, state
FROM customer
WHERE state = "CA"
```

Mehrere Prädikate müssen durch einen der logischen Operatoren OR oder AND getrennt werden. Jedes Prädikat kann durch den Operator NOT negiert werden. Klammern können verwendet werden, um logische Vergleiche und Gruppen von Vergleichen zu isolieren. Dadurch werden unterschiedliche Zeilenevaluierungskriterien produziert. So entnimmt beispielsweise die folgende SELECT-Anweisung alle Zeilen, für welche die Spalte STATE den Wert "CA" enthält, sowie diejenigen mit dem Wert "HI":

```
SELECT company, state
FROM customer
WHERE (state = 'CA') OR (state = 'HI')
```

Die folgende SELECT-Anweisung entnimmt alle Zeilen, in denen die Spalte SHAPE die Werte "round" oder "square" und die Spalte COLOR den Wert „red" enthält. Eine Zeile, die beispielsweise in der Spalte SHAPE der Wert „round" und in der Spalte COLOR der Wert „blue" enthält, würde nicht entnommen werden.

```
SELECT shape, color, cost
```

```
FROM objects
WHERE ((shape = "round") OR (shape = "square")) AND (color = "red")
```

Ohne die Klammern, durch die die Rangfolge der logischen Operatoren verändert wird, erhalten Sie ein völlig anderes Ergebnis (siehe die folgende Anweisung). Diese Anweisung entnimmt die Zeilen, die in der Spalte SHAPE "round" enthalten, unabhängig vom in der Spalte COLOR vorhandenen Wert. Desweiteren werden Zeilen entnommen, die in der Spalte SHAPE den Wert "square" und in der Spalte COLOR den Wert "red" enthalten. Im Gegensatz zum obigen Beispiel würden hier Zeilen entnommen, die in SHAPE "round" und in COLOR "blue" enthalten.

```
SELECT shape, color, cost
FROM objects
WHERE shape = "round" OR shape = "square" AND color = "red"
```

Hinweis

Eine WHERE-Klausel filtert Daten vor der Gruppierung einer GROUP BY-Klausel. Verwenden Sie zum Filtern basierend auf aggregierten Werten eine HAVING-Klausel.

Anwendung

SELECT, UPDATE, DELETE

5.4.3.10 Allgemeine Funktionen und Operatoren

Hier ist eine Liste von Funktionen und Operatoren, die in *TurboSQL* Ausdrücken verwendet werden können. Die Liste setzt sich aus wenigen Standard SQL Funktion und vielen zusätzlichen *TurboDB* Funktionen zusammen.

=

Syntax

```
expr1 = expr2
```

Beschreibung

Test auf Gleichheit.

<

Syntax

```
expr1 < expr2
```

Beschreibung

Test ob der Ausdruck *expr1* kleiner als *expr2* ist.

<=

Syntax

```
expr1 <= expr2
```

Beschreibung

Test ob der Ausdruck expression *expr1* kleiner oder gleich *expr2* ist.

>

Syntax

```
expr1 > expr2
```

Beschreibung

Test ob der Ausdruck *expr1* größer als *expr2* ist.

>=

Syntax

```
expr1 >= expr2
```

Beschreibung

Test ob der Ausdruck *expr1* größer oder gleich *expr2* ist.

BETWEEN ... AND ...

Syntax

```
expr1 BETWEEN expr2 AND expr3
```

Beschreibung

Test ob der Ausdruck *expr1* größer oder gleich *expr2* und kleiner oder gleich *expr3* ist.

IN

Syntax

```
expr IN (expr1, expr2, expr3, ...)
```

Beschreibung

Test ob *expr* mit einem der Ausdrücke *expr1*, *expr2*, *expr3*, ... übereinstimmt.

AND

Syntax

```
cond1 AND cond2
```

Beschreibung

Test ob sowohl *cond1* als auch *cond2* wahr ist.

OR

Syntax

```
cond1 OR cond2
```

Beschreibung

Test ob *cond1* oder *cond2* wahr ist.

NOT

Syntax

```
NOT cond
```

Beschreibung

Test ob *cond* falsch ist.

CASE

Syntax

```
CASE
  WHEN cond1 THEN expr1
  WHEN cond2 THEN expr2
  ...
  [ELSE exprN]
END
CASE expr
  WHEN exprA1 THEN exprB1
  WHEN exprA2 THEN exprB2
  ...
  [ELSE exprBN]
END
```

Beschreibung

Die erste Form der case Operation ermittelt den ersten Ausdruck für den die Bedingung wahr ist. Die zweite liefert den B Ausdruck, dessen A Ausdruck identisch ist zu *expr*.

Beispiele

```
CASE WHEN Age < 8 THEN 'infant'WHEN Age < 18 THEN 'teenager' WHEN Age < 30
THEN 'twen' ELSE 'adult' END
CASE Status WHEN 0 THEN 'OK' WHEN 1 THEN 'WARNING' WHEN 2 THEN 'ERROR' END
```

CAST

Syntax

```
CAST(value AS type)
```

Beschreibung

Wandelt *value* falls möglich in den angegebenen Datentyp *type* um. Die Cast Operation kann Zeichenketten abschneiden und die Genauigkeit von Zahlen vermindern. Falls eine Umwandlung nicht möglich ist, führt CAST zu einem Fehler.

Beispiele

```
CAST(time AS CHAR(10)) --Converts the time in its string representation
CAST(time AS CHAR(3)) --Displays only the first three characters
CAST(amount AS INTEGER)) --Looses the digits after the decimal point
CAST('abc' AS BIGINT) --Raises a conversion error
CAST(34515 AS BYTE) --Raises an overflow error
```

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
TurboPL Functionen und Operatoren

5.4.3.11 Arithmetische Funktionen und Operatoren

Dies ist eine Liste arithmetischer Funktionen und Operatoren, die in TurboSQL verwendet werden können.

+

Syntax

```
value1 + value2
```

Beschreibung

Berechnet die Summe beider Werte.

-

Syntax

```
value1 - value2
```

Beschreibung

Berechnet die Differenz beider Werte.

*

Syntax

```
value1 * value2
```

Beschreibung

berechnet das Produkt beider Werte.

/

Syntax

```
value1 / value2
```

Beschreibung

Berechnet den Quotienten aus beiden Werten.

%

Syntax

```
value1 % value2
```

Beschreibung

Berechnet den Modulo-Operator zweier integraler Zahlen. Nur in TurboDB Managed verfügbar.

ARCTAN

Syntax

```
ARCTAN(value)
```

Beschreibung

Berechnet den Arcus Tangens von value

CEILING

Syntax

```
CEILING(value)
```

Beschreibung

Berechnet die kleinste ganze Zahl größer oder gleich dem gegebenen Wert.

Beispiel

```
CEILING(-3.8) --liefert -3.0
CEILING(3.8) --liefert 4.0
```

COS

Syntax

```
COS(value)
```

Beschreibung

Berechnet den Cosinus von value

DIV

Syntax

```
a div b
```

Beschreibung

Ganzzahlige Division

Beispiel

```
35 div 6 --returns 5
-35 div 6 --returns -5
35 div -6 --returns -5
-35 div -6 --returns 5
```

EXP

Syntax

```
EXP(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet das Exponential von value (zur Basis e)

FLOOR

Syntax

```
FLOOR(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet den größten ganzzahligen Wert kleiner oder gleich dem gegebenen Wert.

Beispiel

```
FLOOR(-3.8) --liefert -4.0
FLOOR(3.8) --liefert 3.0
```

FRAC

Syntax

```
FRAC(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Errechnet das Bruchteil der realen Zahl.

Beispiel

```
FRAC(-3.8) --returns -0.8
FRAC(3.8) --returns 0.8
```

INT

Syntax

```
INT(:X DOUBLE) RETURNS BIGINT
```

Beschreibung

Errechnet den ganzzahligen Bestandteil einer realen Zahl als ganze Zahl

Beispiel

```
INT(-3.8) --returns -3
INT(3.8) --returns 3
```

LOG

Syntax

```
LOG(:X DOUBLE) RETURNS BIGINT
```

Beschreibung

Errechnet den natürlichen Logarithmus einer realen Zahl.

MOD

Syntax

```
a mod b
```

Beschreibung

Errechnet den Rest einer ganzzahligen Division. Es gilt *a mod b = a - (a div b) * b*.

Beispiel

```
35 mod 6 --returns 5
35 mod -6 --returns 5
-35 mod 6 --returns 5
-35 mod -6 --returns 5
```

ROUND

Syntax

```
ROUND(:X DOUBLE [, :Precision BYTE]) RETURNS DOUBLE
```

Beschreibung

Rundet *value* zu der gegebenen Anzahl an Stellen.

Nur in TurboDB Native verfügbar.

Beispiel

```
ROUND(3.141592, 3) --liefert 3.142
```

SIN

Syntax

```
SIN(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet den Sinus von value

SQRT

Syntax

```
SQRT(:X DOUBLE) RETURNS DOUBLE
```

Beschreibung

Berechnet die Quadratwurzel von value

See also

General Functions and Operators
Arithmetic Functions and Operators
String Functions and Operators
Date and Time Functions and Operators
Aggregation Functions
Miscellaneous Functions and Operators
TurboPL Functions and Operators

5.4.3.12 Zeichenketten Funktionen und Operatoren

Dies ist eine Liste von Funktionen und Operatoren zur Bearbeitung von Zeichenketten, die in TurboSQL eingesetzt werden können.

||

Syntax

```
string1 || string2
```

Beschreibung

Verknüpft die beiden Zeichenketten.

ASCII

Syntax

```
ASCII(string)
```

Beschreibung

Berechnet den Code des ersten Zeichens in der Zeichenkette. Liefert NULL falls das Argument NULL oder leer ist.

CHAR_LENGTH

Syntax

```
CHAR_LENGTH(string)
```

Beschreibung

Berechnet die Anzahl der Zeichen in der Zeichenkette.

HEXSTR

Syntax

```
HEXSTR(number, width)
```

Beschreibung

Berechnet ein hexadezimale Darstellung der Zahl mit mindestens *width* Zeichen

Beispiel

```
HEXSTR(15, 3) --returns '00F'
```

LEFTSTR

Syntax

```
LEFTSTR(string, count)
```

Beschreibung

Liefert die ersten *count* Zeichen der übergebenen Zeichenkette *string*.

LEN

Syntax

```
LEN(string)
```

Beschreibung

Wie *CHAR_LENGTH*. *CHAR_LENGTH* als Standard SQL ist hier vorzuziehen.

LIKE

Syntax

```
string1 [NOT] LIKE string2
```

Beschreibung

Vergleicht die beiden Zeichenketten wie in Standard SQL definiert. Als Joker Zeichen kann % und _ verwendet werden.

Beispiele

```
'Woolfe' LIKE 'Woo%'
Name LIKE '_oolfe'
```

LOWER

Syntax

```
LOWER(string)
```

Beschreibung

Liefert den String in Kleinbuchstaben.

RIGHTSTR

Syntax

```
RIGHTSTR(string, count)
```

Beschreibung

Liefert die letzten *count* Zeichen der gegebenen Zeichenkette *string*.

STR

Syntax

```
STR(number, width, scale, thousand_separator, fill_character,
decimal_separator)
STR(enumeration_column_reference)
```

Beschreibung

Die erste Variante liefert eine Darstellung der Zahl als Zeichenkette mit den angegebenen Formatierung.

Die zweite Variante liefert eine Darstellung des Aufzählungs-Werts als Zeichenkette.

Beispiel

```
STR(3.14159, 10, 4, ',', '*', '.') --returns ****3.1416
```

SUBSTRING

Syntax

```
SUBSTRING(string FROM start [FOR length])
```

Beschreibung

Liefert den Teilstring der Länge *length* aus *string* angefangen mit dem Zeichen an Position *start*

TRIM

Syntax

```
TRIM([kind [char] FROM] string)
```

Beschreibung

Liefert einen String ohne führende oder abschließende Zeichen.
kind ist eines dieser Schlüsselwörter: LEADING, TRAILING, BOTH

char ist das Zeichen, das entfernt wird. Wird nichts angegeben, werden Leerzeichen entfernt.

Beispiele

die folgenden Ausdrücke liefern alle 'Carl':

```
TRIM('  Carl      ')
TRIM(LEADING FROM '   Carl')
TRIM(TRAILING FROM 'Carl    ')
TRIM(BOTH 'x' FROM 'xxCarlxx')
```

UPPER

Syntax

```
UPPER(string)
```

Beschreibung

Liefert den String in Großbuchstaben.

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
TurboPL Functionen und Operatoren

5.4.3.13 Datum und Zeit Funktionen und Operatoren

Dies ist eine Liste von Datum und Zeit Funktionen und Operatoren, die in TurboSQL benützt werden können.

+

Syntax

```
date + days
timestamp + days
time + minutes
```

Beschreibung

Addiert eine Anzahl an Tagen zu einem Datum oder einem Zeitstempel. Addiert eine Anzahl an Minuten zu einem Zeitwert.

Beispiele

```
CURRENT_DATE + 1 --Das morgige Datum
CURRENT_TIMESTAMP + 1 --Das morgige Datum mit der aktuellen Zeit
CURRENT_TME + 60 --In einer Stunde
CURRENT_TIME + 0.25 --15 Sekunden später
```

-

Syntax

```
date - days
date1 - date2
timestamp - days
timestamp1 - timestamp2
time - minutes
time1 - time2
```

Beschreibung

Subtrahiert eine Anzahl an Tagen von einem Datum oder Zeitstempel. Subtrahiert eine Anzahl an Minuten von einem Zeit-Wert. Berechnet die Anzahl an Tagen zwischen zwei Datum- oder Zeitstempel-Werten. Berechnet die Anzahl an Minuten zwischen zwei Zeiten.

Beispiele

```
CURRENT_DATE - 1 --Gestern
CURRENT_TIMESTAMP - 1 --Vor 24 Stunden
CURRENT_DATE - DATE'1/1/2006' --Anzahl der Tage seit Anfang 2006
```

```
CURRENT_TIME - 60 --Vor einer Stunde
CURRENT_TIME - TIME'12:00 pm' --Anzahl der Stunden seit Mittag (das kann auch
negativ sein)
```

CURRENT_DATE

Syntax

```
CURRENT_DATE
```

Beschreibung

Liefert das aktuelle Datum.

CURRENT_TIME

Syntax

```
CURRENT_TIME
```

Beschreibung

Liefert die aktuelle Zeit auf die Millisekunde genau.

CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
```

Beschreibung

Liefert den aktuellen Zeitstempel mit einer Genauigkeit von einer Millisekunde. (d.h. *CURRENT_DATE* und *CURRENT_TIME* zusammen)

DATETIMESTR

Syntax

```
DATETIMESTR(TimeStamp, Precision)
```

Beschreibung

Liefert den gegebenen Zeitstempel-Wert als Zeichenkette im Format der aktuellen Ländereinstellung. Die Genauigkeit *Precision* ist 2 für Minuten, 3 für Sekunden und 4 für Millisekunden.

EXTRACT

Syntax

```
EXTRACT(kind FROM date)
```

Beschreibung

Berechnet einen Wert aus *date*. *kind* ist eines der folgenden Schlüsselwörter:

YEAR	Liefert das Jahr.
MONTH	Liefert den Monat.
DAY	Liefert den Tag.
WEEKDAY	Liefert den Wochentag, mit Montag als 1, Dienstag 2 usw.
WEEKDAYNAME	Liefert den Namen des Wochentages in der aktuellen Spracheinstellung.
WEEK	Liefert die Wochennummer wie im ISO standard festgelegt.
HOUR	Liefert die Stunde.
MINUTE	Liefert die Minute.
SECOND	Liefert die Sekunde.
MILLISECOND	Liefert die Millisekunde.

Beispiele

```
EXTRACT(DAY FROM CURRENT_DATE)
EXTRACT(HOUR FROM CURRENT_TIME)
```

```
EXTRACT(SECOND FROM CURRENT_TIMESTAMP)
EXTRACT(WEEKDAYNAME FROM CURRENT_DATE)
EXTRACT(MILLISECOND FROM CURRENT_TIME)
EXTRACT(WEEK FROM CURRENT_TIMESTAMP)
```

MAKEDATE

Syntax

```
MAKEDATE(year, month, day)
```

Beschreibung

Liefert den Datum-Wert für das gegebene Datum.

Beispiele

```
SELECT * FROM MyTable WHERE Abs(Today - MakeDate(EXTRACT(YEAR FROM
CURRENT_DATE), EXTRACT(MONTH FROM Birthday), EXTRACT(DAY FROM Birthday))) < 7
```

MAKETIMESTAMP

Syntax

```
MAKETIMESTAMP(year, month, day, hour, minute, second, millisecond)
```

Beschreibung

Liefert den Zeitstempel-Wert für das gegebene Datum und die gegebene Zeit.

MAKETIME

Syntax

```
MAKETIME(hour, minute, second, millisecond)
```

Beschreibung

Liefert den Zeit-Wert für die angegebene Zeit.

TIMESTR

Syntax

```
TIMESTR(time, precision)
```

Beschreibung

Liefert die gegebene Zeit als Zeichenkette gemäß der aktuellen Ländereinstellung. Die Genauigkeit *precision* ist 2 für Minuten, 3 für Sekunden und 4 für Millisekunden.

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
TurboPL Functionen und Operatoren

5.4.3.14 Aggregat Funktionen

Dies ist eine Liste von Aggregat Funktionen, die in TurboSQL verwendet werden können.

AVG

Syntax

```
AVG(column_reference)
```

Beschreibung

Berechnet den Durchschnitt der Werte in der Spalte

COUNT

Syntax

```
COUNT(*|column_reference)
```

Beschreibung

Berechnet die Anzahl der Zeilen in der Spalte

Beispiele

```
COUNT(*)
COUNT(NAME)
```

MAX

Syntax

```
MAX(column_reference)
```

Beschreibung

Berechnet das Maximum der Werte in der Spalte

MIN

Syntax

```
MIN(column_reference)
```

Beschreibung

Berechnet das Maximum der Werte in der Spalte

STDDEV

Syntax

```
STDDEV(column_reference)
```

Beschreibung

Berechnet die Standardabweichung der Werte in den Spalten. Das Argument muss numerisch sein. Das Ergebnis ist immer ein FLOAT.

Beispiel

```
SELECT AVG(Value), STDDEV(Value) FROM Values
```

SUM

Syntax

```
SUM(column_reference)
```

Beschreibung

Berechnet die Summe der Werte in der Spalte

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
TurboPL Functionen und Operatoren

5.4.3.15 Sonstige Funktionen und Operatoren

Dies ist eine Liste sonstiger Funktionen zum Einsatz in *TurboSQL*.

CONTAINS

Syntax

```
CONTAINS(full-text-search-expression IN table-name.*)
```

Beschreibung

Liefert für jeden Datensatz true, der dem Volltext-Suchausdruck genügt.

Diese Funktionen gibt es nicht in TurboDB Managed.

NEWGUID

Syntax

```
NEWGUID
```

Beschreibung

Erzeugt einen neuen Globally Unique Identifier, wie zum Beispiel {2A189230-2041-44A6-87B6-0AFEE240F09E}.

Beispiel

```
INSERT INTO TABLEA ("Guid") VALUES(NEWGUID)
```

CURRENTRECORDID

Syntax

```
CURRENTRECORDID(table_name)
```

Beschreibung

Liefert den zuletzt verwendeten AutoInc-Wert des gegebenen Tabelle. Mit dieser Funktion ist es möglich, durch mehrere Kommandos innerhalb eines Statements verknüpfte Datensätze in mehrere Tabellen einzutragen.

NULLIF

Syntax

```
NULLIF(value1, value2)
```

Beschreibung

Liefert NULL falls *value1* und *value2* gleich sind und *value1* falls nicht.

Nur in TurboDB Managed.

Siehe auch

Allgemeine Funktionen und Operatoren
Arithmetische Funktionen und Operatoren
String Funktionen und Operatoren
Datum und Zeit Funktionen und Operatoren
Aggregat Funktionen
Sonstige Funktionen und Operatoren
TurboPL Functionen und Operatoren

5.4.3.16 Tabellen Operatoren

TurboSQL unterstützt die folgenden Operatoren um Tabellen zu kombinieren. Sie genügen alle der Standard SQL Spezifikation:

JOIN

Syntax

```
table_reference [INNER | LEFT OUTER | RIGHT OUTER | OUTER] JOIN
table_reference
```

Beispiel

```
SELECT * FROM A JOIN B ON A.a = B.a
SELECT * FROM A LEFT OUTER JOIN B ON A.a = B.a
```

Beschreibung

Liefert alle Zeilenpaare der beiden Tabellenreferenzen, für die die Bedingung gilt.

UNION

Syntax

```
table_term UNION [ALL] table_term [CORRESPONDING BY column_list]
```

Beispiel

```
TABLE A UNION TABLE B
```

Beschreibung

Liefert alle Zeilen aus den beiden Tabelle. Das Ergebnis ist eindeutig, falls *ALL* nicht definiert ist. Die beiden Tabellen müssen über kompatible Spalten verfügen.

EXCEPT

Syntax

```
table_term EXCEPT [ALL] table_term CORRESPPONDING [BY column_list]
```

Beispiel

```
SELECT * FROM TABLE A EXCEPT SELECT * FROM TABLE B
```

Beschreibung

Liefert alle Zeilen der ersten Tabelle, die nicht in der zweiten Tabelle enthalten sind. Die Ergebnismenge ist eindeutig, falls *ALL* nicht definiert ist. Die beiden Tabellen müssen über kompatible Spalten verfügen.

INTERSECT

Syntax

```
table_primitive INTERSECT table_primitive CORRESPONDING [BY column_list]
```

Beispiel

```
SELECT * FROM TABLE A INTERSECT [ALL] SELECT * FROM TABLE B
```

Beschreibung

Liefert alle Zeilen, die sowohl in der ersten als auch in der zweiten Tabelle enthalten sind. Das Ergebnis ist eindeutig, falls *ALL* nicht definiert ist. Die beiden Tabellen müssen über kompatible Spalten verfügen.

5.4.3.17 Unterabfragen

Suchbedingungen in SELECT, INSERT und UPDATE Statements können eingebettete Abfragen beinhalten, die mit einem der folgenden Operatoren mit der Hauptabfrage verglichen werden können. Außerdem kann eine geklammerte Unterabfrage überall da verwendet werden, wo ein Ausdruck erwartet wird. TurboSQL erlaubt sowohl korrelierte als auch unkorrelierte Unterabfragen.

In

Prüfung ob der Wert eines Ausdrucks in der Ergebnismenge der Unterabfrage gefunden werden kann.

Beispiel

```
select * from SALESINFO
where customerName in (
  select name from CUSTOMER where state = 'CA'
)
```

Liefert alle Verkäufe an Kunden aus Kalifornien und ist im Allgemeinen dasselbe wie

```
select * from SALESINFO join CUSTOMER on customerName = name
where state = 'CA'
```

Exists

Prüft ob die Unterabfrage mindesten einen Datensatz enthält

Example

```
select * from SALESINFO
where exists (
  select * from CUSTOMER
  where name = SALESINFO.customerName and state = 'CA'
)
```

Liefert dasselbe Ergebnis wie das erste Beispiel. Zu beachten ist aber, dass die Unterabfrage jetzt eine Spaltenreferenz auf die äußere Abfrage enthält. Dies wird als korrelierte Unterabfrage bezeichnet.

Any/Some

Überprüft, ob es mindestens einen Datensatz in der Ergebnismenge der Unterabfrage gibt, der die Suchbedingung erfüllt.

Example

```
select * from SALESINFO
where amount > any (
  select averageAmount from CUSTOMER
  where name = SALESINFO.customerName
)
```

Liefert alle Verkäufe, die für den jeweiligen Kunden größer als der Durchschnitt sind.

All

Überprüft, ob alle Datensätze der Ergebnismenge der Unterabfrage der Suchbedingung genügen.

Example

```
select * from SALESINFO
where amount > all (
  select averageAmount from CUSTOMER
  where state = 'CA'
)
```

Liefert die Verkäufe, die für jeden einzelnen Kunden in Kalifornien größer als der Durchschnitt sind.

Unterabfrage als Ausdruck

Eine Unterabfrage in Klammern kann also skalarer Ausdruck verwendet werden. Der Typ des Ausdrucks ist der Typ der ersten Spalte der Ergebnismenge. Der Wert des skalaren Ausdrucks ist Wert der ersten Spalte in der ersten Zeile. Wenn die Ergebnismenge keine Spalte enthält, ist der Ausdruck ungültig. Enthält sie keine Zeile, ist der Wert NULL.

Beispiele:

```
select * from [TableB] where C1 like (select C2 from TableB) || '%'
set A = (select Count(*) from TableA)
```

Verfügbarkeit:

Die Verwendung von Unterabfragen als Ausdrücke ist nur in TurboDB Managed verfügbar.

Siehe auch

WHERE

5.4.3.18 Volltextsuche

Volltextsuche ist die Suche nach einem beliebigen Wort in einem Datensatz. Diese Art der Suche ist für Memo- und WideMemo-Felder besonders nützlich, in denen das Suchen mit herkömmlichen Operatoren und Funktionen nicht das erwartete Resultat liefert oder zu lange dauert.

Die Volltextsuche unterliegt zwei Einschränkungen:

- Es muss einen Volltext-Index für die Tabelle geben.
- Eine Volltextsuche bezieht sich immer auf genau eine Tabelle (Es können aber mehrere Volltext-Suchbedingungen in einer WHERE-Klausel sein.).

Die Basis eines Volltext-Index ist das Wörterbuch, das eine normale Datenbank-Tabelle mit einem bestimmten Schema ist. Es enthält die Informationen über indizierte Wörter, ausgeschlossene Wörter, Wort-Relevanz, u.s.w. Sobald das Wörterbuch besteht, kann es für eine beliebige Zahl von Volltext-Indexen auf einer oder auf mehreren Tabellen benutzt werden.

Ab TurboDB 5, werden Volltext-Suchbedingungen in die WHERE Klausel der Abfrage eingebettet:

```
select * from SOFTWARE join VENDOR on SOFTWARE.VendorId = VENDOR.Id
where VENDOR.Country = 'USA' and (contains('office' in SOFTWARE.*) or contains
('Minneapolis' in VENDOR.*))
```

Eine einfache Volltext-Suchbedingung sieht so aus:

```
contains('office -microsoft' in SOFTWARE.*)
```

die zutreffend ist, wenn irgendein Feld des Standard-Volltext-Index der Tabelle SOFTWARE das Wort *office* aber nicht das Wort *microsoft* enthält. Wenn sich die Abfrage auf nur eine Tabelle bezieht, kann dieses auch so geschrieben werden:

```
contains('office -microsoft' in *)
```

Wenn der Volltext-Suchausdruck mehr als ein Wort ohne den Bindestrich enthält, sucht TurboDB nach Datensätzen, die alle gegebenen Wörter enthalten.

```
contains('office microsoft' in SOFTWARE.*)
```

wird daher Datensätze finden, die beide Wörter, *office* und *microsoft,* in einem Feld des Standard-Volltext-Index der Tabelle enthalten.

Nach Wörter, die durch ein Pluszeichen getrennt sind, wird alternativ gesucht. Das Prädikat

```
contains('office star + open' in SOFTWARE.*)
```

findet Datensätze, die das Wort *office* und entweder *star* oder *open* beinhalten (oder beide).

Hinweise

Volltext-Indexe können mit der TurboSQL Anweisung CREATE FULLTEXTINDEX , mit einem der Datenbank-Management Werkzeuge (wie TurboDB5Viewer) oder mit den entsprechenden Methoden der jeweiligen TurboDB Komponenten erstellt werden.

Von TurboDB Tabellen-Level 3 auf Level 4 hat sich die Technologie der Volltext-Indizierung geändert. Die neue Implementierung ist sehr viel schneller und erlaubt sowohl gewartete Indexe als auch Relevanzen. Es wird dringend empfohlen Tabellen-Level 4 zu verwenden, wenn mit Volltext-Indexen gearbeitet werden soll. Die alte Volltextsuche wird eventuell entfernt.

5.4.4 Data Definition Language

Turbo SQL bietet die folgenden Kommandos und Datentypen als Teil der DDL:

Kommandos

CREATE TABLE

ALTER TABLE

CREATE INDEX

CREATE FULLTEXTINDEX

UPDATE INDEX/FULLTEXTINDEX

DROP

Datentypen

TurboSQL Datentypen

5.4.4.1 CREATE TABLE Befehl

Erzeugt eine neue Tabelle für die aktuelle Datenbank.

Syntax

```
CREATE TABLE table_reference
[LEVEL level_number]
[ENCRYPTION encryption_algorithm]
[PASSWORD password]
[LANGUAGE language]
(column_definition | constraint_definition [, column_definition |
constraint_definition] ...)
```

wobei eine eine Spaltendefinition *column_definition* so aussieht,

```
column_reference data_type [NOT NULL] [DEFAULT expression]
```

(siehe Datentypen für Tabellenspalten für weitere Informationen)

und eine Gültigkeitsbedingung *constraint_definition* so:

```
PRIMARY KEY (column_reference [, column_reference]...) |
UNIQUE (column_reference [, column_reference]...) |
[CONSTRAINT constraint_name] CHECK (search_condition) |
FOREIGN KEY (column_reference [, column_reference]...)
REFERENCES table_reference (column_reference [, column_reference]...)
[ON UPDATE NO ACTION | CASCADE]
```

```
[ON DELETE NO ACTION | CASCADE]
```

Beschreibung

Verwenden Sie CREATE TABLE wenn Sie eine neue Tabelle zur aktuellen Datenbank hinzufügen möchten.

```
CREATE TABLE MyTable (
  OrderNo AUTOINC,
  OrderId CHAR(20) NOT NULL,
  Customer LINK("Customer") NOT NULL,
  OrderDate DATE NOT NULL DEFAULT Today,
  Destination ENUM("Home", "Office", "PostBox"),
  PRIMARY KEY (OrderNo),
  CHECK (LEN(OrderId) > 3),
  FOREIGN KEY (Customer) REFERENCES Customer (CustNo) ON DELETE CASCADE ON
UPDATE NO ACTION
)
```

Um ein Passwort, einen Schlüssel und eine Sprache zu definieren, fügen Sie die entsprechenden Schlüsselwörter hinzu:

```
CREATE TABLE MyTable
  ENCRYPTION Blowfish PASSWORD 'u(i,iUklah'
  LANGUAGE 'ENU' (Name CHAR(20)
)
```

Die unterstützten Verschlüsselungs-Algorithmen sind in "Data Security" beschrieben.
Der Level kann angegeben werden um rückwärts kompatibel zu bleiben. Falls kein Level angegeben wird, wird das aktuellste Format erzeugt.

Die anderen Klauseln haben ihre Standard-SQL-Bedeutung. Zu beachten ist, daß TurboSQL noch nicht die set null und set default Aktionen für Fremdschlüssel unterstützt, die im SQL-Standard vorhanden ist.

Siehe auch

Spalten Datentypen

5.4.4.2 ALTER TABLE Befehl

Fügt einer bestehenden Tabelle Spalten hinzu und löscht Spalten aus ihr.

Syntax

```
ALTER TABLE table_reference
[LEVEL level_number]
[ENCRYPTION encryption_algorithm]
[PASSWORD password]
[LANGUAGE language]
DROP column_reference |
DROP constraint_name |
ADD column_definition |
ADD constraint_definition |
RENAME column_reference TO column_reference |
MODIFY column_definition
```

Beschreibung

Verwenden Sie die Anweisung ALTER TABLE, um das Schema einer bestehenden Tabelle zu ändern. Die Beschreibungen für column_definition and constraint_definition lesen Sie bitte in CREATE TABLE Befehl. Es gibt sechs verschiedene Optionen:

Löschen einer exitierenden Spalte mit DROP:

```
ALTER TABLE Orders DROP Destination
```

column_reference muss sich auf eine existierende Spalte beziehen. Turbo SQL Spaltenbezeichner unterscheiden zwischen Groß- und Kleinschreibung.

Löschen einer existierenden Gültigkeitsbedingung:

```
ALTER TABLE Orders DROP PrimaryKey
```

Hinzufügen einer neuen Spalte mit ADD:

```
ALTER TABLE Orders ADD Date_of_delivery DATE
```

Der Name der neuen Spalte darf in der Tabelle noch nicht vorkommen.

Hinzufügen einer neuen Gültigkeitsbedingung mit ADD:

```
ALTER TABLE Orders ADD CONSTRAINT RecentDateConstraint CHECK (Date_of_delivery
> 1.1.2000)
ALTER TABLE Orders ADD FOREIGN KEY (Customer) REFERENCES Customer (CustNo)
```

Ändern des Namens einer existierenden Spalte mit RENAME:

```
ALTER TABLE Orders RENAME Date_of_delivery TO DateOfDelivery
```

Die erste *column_reference* ist der Name der existierenden Spalte, die zweite ist der neue Name für diese Spalte. Umbenennen einer Spalte verändert die enthaltenen Daten nicht.

Ändern des Datentyps einer existierenden Spalte mit MODIFY:

```
ALTER TABLE Orders MODIFY DateOfDelivery TIMESTAMP
```

column_reference muss sich auf eine existierende Spalte beziehen. Der Typ der Spalte kann in einen der verfügbaren Datentypen konvertiert werden. Die enthaltenen Daten werden erhalten, falls möglich.

Die Parameter *level_number*, *password*, *key* und *language* haben dieselbe Bedeutung wie in CREATE TABLE. Falls *password* und *key* weggelassen werden, bleiben aktuellen Einstellungen erhalten. Um die Verschlüsselung aufzuheben, ist sie auf NONE zu setzen.

Diese Anweisung hebt die Verschlüsselung auf:

```
ALTER TABLE Orders ENCRYPTION None
```

Es ist möglich, mehrere Änderungen in beliebiger Reihenfolge in der selben Anweisung zu kombinieren:

```
ALTER TABLE Orders
  ADD Date_of_delivery DATE,
  DROP Destination,
  ADD DeliveryAddress CHAR(200),
  RENAME Customer TO CustomerRef
```

Anmerkung: RENAME und MODIFY sind proprietäre Erweiterungen zu SQL-92.

Kompatibilität: TurboDB Managed unterstützt zur Zeit nur die ADD (column) und DROP COLUMN Klauseln.

Siehe auch

Spalten datentypen

5.4.4.3 CREATE INDEX Befehl

Erzeugt einen neuen Index für eine bestehende Tabelle.

Syntax

```
CREATE [UNIQUE] INDEX index_reference ON table_reference (column_reference
[ASC|DESC] [,column_reference [ASC|DESC] ...] )
```

Beschreibung

Indexe werden verwendet um Suchaktionen zu beschleunigen. Verwenden Sie CREATE INDEX um einen neuen Index für eine bestehende Tabelle zu erstellen:

```
CREATE INDEX OrderIdIdx ON Orders (OrderId ASC)
```

Sie können auch mehrstufige hierarchische Indexe erstellen:

```
CREATE INDEX TargetDateIdx ON Orders (DeliveryDate DESC, OrderId)
```

Verwenden Sie UNIQUE um einen Index zu definieren, der einen Fehler erzeugt, falls nicht eindeutige Spaltenwerte in die Tabelle geschrieben werden sollen. Die Voreinstellung ist nicht eindeutig.

Anmerkung

Die Attribute ASC und DESC sind properietäre Erweiterungen des SQL-92 Standards.

5.4.4.4 CREATE FULLTEXTINDEX Statement

Erstelle einen neuen Volltext-Index für eine bestehende Tabelle.

Syntax

```
CREATE FULLTEXTINDEX index_reference ON table_reference (column_reference [,
column_reference ...]) DICTIONARY table_reference [CREATE] [UPDATE]
```

Beschreibung

Ein Volltext-Index ermöglicht eine Suche mit Volltext-Suchbedingungen wie dem CONTAINS Prädikat. Die Spaltenreferenzen sind eine Liste der Tabellenspalten aller Datentypen mit Ausnahme von Blobs, einschließlich Memos und WideMemos. Volltext-Index benötigen eine zusätzliche Datenbank-Tabelle, die eine Liste der indizierten Wörter enthält, das Wörterbuch.

Die Wörterbuch-Tabelle kann mit diesem Statement oder explizit erstellt werden. Falls CREATE nicht verwendet wird, erwartet das Statement eine Wörterbuch-Tabelle, mit den folgenden Eigenschaften:

- Die erste Spalte ist vom Typ *VARCHAR* oder *VARWCHAR* beliebiger Länge. Diese Spalte nimmt die möglichen Suchbegriffe auf. Falls ein Wort länger ist als es diese Spalte erlaubt, wird es abgeschnitten.
- Die zweite Spalte ist vom Typ *BYTE*. Es nimmt die globale Relevanz des Wortes auf.
- Weitere Spalten können entsprechend den Notwendigkeiten der Anwendung folgen.
- Es muss eine Spalte vom Typ *AUTOINC* geben, um die Wörter zu identifizieren. Die Anzeigeinformation dieser *AUTOINC* Spalte muss die erste Spalte der Tabelle sein.

Falls die CREATE Klausel angegeben wird, erstelle das Statement eine neue Wörterbuch-Tabelle mit einer ersten Spalte vom Typ *VARCHAR(20)*.

Falls UPDATE verwendet wird, werden Worte, die in der Wörterbuch-Tabelle nicht gefunden werden ergänzt. Falls UPDATE nicht angegeben ist, werden nur Wörter der Wörterbuch-Tabelle indiziert und können bei Suchen gefunden werden.

Hinweis

Die Technologie der Volltext-Suche für Tabellen bis Tabellen-Level 3 unterscheidet sich von der ab -Level 4. Erst ab Tabellen-Level 4 wird automatische Wartung und Berechnung der Relevanz unterstützt.

5.4.4.5 DROP Command

Eine Tabelle oder einen Index aus der Datenbank entfernen.

Syntax

```
DROP TABLE table_reference
DROP INDEX table_reference.index_name
DROP FULLTEXTINDEX table_reference.index_name
```

Beschreibung

Verwenden Sie DROP um ein Datenbank-Objekt und alle seine zugehörigen Dateien komplett aus der Datenbank zu entfernen.

```
DROP INDEX Orders.OrderIdIdx
DROP TABLE Orders
```

Achtung

Während ein versehentlich gelöschter Index einfach wiederhergestellt werden kann, sind die Daten einer Tabelle nach DROP unwiederbringbar verloren.

5.4.4.6 UPDATE INDEX/FULLTEXTINDEX Statement

Repariert einen Index oder Volltext-Index.

Syntax

```
UPDATE INDEX table_reference.index_name
UPDATE FULLTEXTINDEX table_reference.index_name
```

Beschreibung

Falls ein Index nicht mehr aktuell oder kaputt ist (das kann in der dateibasierten Version von TurboDB passieren, wenn die Anwendung stirbt), kann der Index mit diesem Befehl wiederhergestellt werden. Verwenden Sie den Asterisk * für *index_reference* um alle Indexe der Tabelle wiederherzustellen.

Anmerkung

Die UPDATE FULLTEXTINDEX Anweisung ist nur für den neuen, gewarteten Volltext-Index und dem Tabellen-Level 4 verfügbar.

Beispiel

```
UPDATE INDEX MyTable.*; UPDATE FULLTEXTINDEX MyTable.*
```

5.4.4.7 Datentypen für Tabellenspalten

Die folgenden Spaltentypen werden von *TurboSQL* angeboten:

AUTOINC

Syntax

```
AUTOINC(indication)
```

TurboDB Spaltentyp

AutoInc

Beschreibung

Integer Feld, bezieht einen eindeutigen Wert von der Datenbank-Engine. Anzeige des Feld-Inhaltes gemäß der Index-Beschreibung. (Siehe "Die Automatic Data Link Technologie".)

Beispiel

```
AUTOINC('LastName, FirstName')
```

BIGINT

Syntax

```
BIGINT [NOT NULL]
```

TurboDB Spaltentyp

BigInt

Beschreibung

Ganzzahl zwischen –2^63 und +2^63–1

Beispiel

```
BIGINT DEFAULT 4000000000
```

BIT

Nicht unterstützt in TurboSQL.

BOOLEAN

Syntax

```
BOOLEAN [NOT NULL]
```

TurboDB Spaltentyp

Boolean

Beschreibung

Mögliche Werte sind TRUE und FALSE

Beispiel

```
BOOLEAN DEFAULT TRUE
```

BYTE

Syntax

```
BYTE [NOT NULL]
```

TurboDB Spaltentyp

Byte

Beschreibung

Ganzzahl zwischen 0 und 255

Beispiel

```
BYTE NOT NULL DEFAULT 18
```

CHAR

Syntax

```
CHAR(n) [NOT NULL]
```

TurboDB Spaltentyp

String

Beschreibung

Ansi String bis N Zeichen. 1 <= N <= 255

Beispiel

```
CHAR(40)
```

CURRENCY

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION oder BIGINT.

DATE

Syntax

```
DATE [NOT NULL]
```

TurboDB Spaltentyp

Date

Beschreibung

Datum zwischen 1.1.1 und 31.12.9999

Beispiel

```
DATE
```

DECIMAL

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION.

DOUBLE PRECISION

Syntax

```
DOUBLE [PRECISION][(p)] [NOT NULL]
```

TurboDB Spaltentyp

Float

Beschreibung

Fließkommazahl zwischen 5.0 x 10^–324 und 1.7 x 10^308 mit 15 signifikanten Stellen. P ist die optionale Anzahl an Nachkommastellen für die Anzeige, die zur Konvertierung von Werten in Zeichenketten benutzt wird.

Beispiel

```
DOUBLE(4) NOT NULL
```

ENUM

Syntax

```
ENUM(value1, value2, value3, ...) [NOT NULL]
```

TurboDB Spaltentyp

Enum

Beschreibung

Ein Wert aus der gegebenen Aufzählung. Die Werte müssen gültige Bezeichner mit maximal 40 Zeichen Länge sein. Die Summe der Werte darf 255 Zeichen nicht überschreiten. Es können maximal 16 Werte angegeben werden.

Beispiel

```
ENUM(Red, Blue, Green, Yellow)
```

FLOAT

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION.

GUID

Syntax

```
GUID [NOT NULL]
```

TurboDB Spaltentyp

Guid

Beschreibung

Datentyp für Globally Unique Identifier, 16 Bytes groß

Beispiel

```
GUID DEFAULT '12345678-abcd-abcd-efef-010101010101'
```

INTEGER

Syntax

```
INTEGER [NOT NULL]
```

TurboDB Spaltentyp

Integer

Beschreibung

Ganzzahl zwischen –2.147.483.648 und +2.147.483.647

Beispiel

```
INTEGER NOT NULL
```

LINK

Syntax

```
LINK(table_reference) [NOT NULL]
```

TurboDB Spaltentyp

Link

Beschreibung

Beinhaltet den Wert einer AutoInc-Spalte einer anderen Tabelle und stellt auf diese Weise eine 1:n Beziehung her (Siehe "Die Automatic Data Link Technologie").

Beispiel

```
LINK(PARENTTABLE)
```

LONGVARBINARY

Syntax

```
LONGVARBINARY [NOT NULL]
```

TurboDB Spaltentyp

Blob

Beschreibung

Bit-Stream für beliebige Daten bis 2 GB

Beispiel

```
LONGVARBINARY
```

LONGVARCHAR

Syntax

```
LONGVARCHAR [NOT NULL]
```

TurboDB Spaltentyp

Memo

Beschreibung

Ansi String variabler Länge bis zu 2 G Zeichen

Beispiel

```
LONGVARCHAR
```

LONGVARWCHAR

Syntax

```
LONGVARWCHAR [NOT NULL]
```

TurboDB Spaltentyp

WideMemo

Beschreibung

Unicode String variabler Länge bis zu 2 G Zeichen

Beispiel

```
LONGVARWCHAR
```

MONEY

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION oder BIGINT.

NUMERIC

Nicht unterstützt in *TurboSQL*, verwenden Sie DOUBLE PRECISION.

RELATION

Syntax

```
RELATION(table_reference)
```

TurboDB Spaltentyp

Relation

Beschreibung

Beinhaltet die Werte einer beliebigen Anzahl von AutoInc-Spalten einer anderen Tabelle und stellt auf diese Weise eine m:n Beziehung her. (Siehe "Die Automatic Data Link Technologie").

Kompatibilität

Feature wird in TurboDB Managed 1.x nicht unterstützt.

Beispiel

```
RELATION(PARENTTABLE)
```

TIME

Syntax

```
TIME[(p)] [NOT NULL]
```

TurboDB Spaltentyp

Time

Beschreibung

Tageszeit mit einer Genauigkeit von p, wobei p = 2 für Minuten, p = 3 für Sekunden und p = 4 für Millisekunden steht. Die Voreinstellung ist 3. Die Genauigkeit ist erst ab Tabellenlevel 4 verfügbar. Für niedrigere Level gilt p = 2.

Beispiel

```
TIME(4) DEFAULT 8:32:12.002
```

TIMESTAMP

Syntax

```
TIMESTAMP [NOT NULL]
```

TurboDB Spaltentyp

DateTime

Beschreibung

Kombinierter Datentyp für Datum und Zeit mit einer Genauigkeit von Millisekunden zwischen 1.1.1 00:00:00.000 und 31.12.9999 23:59:59.999

Beispiel

```
TIMESTAMP DEFAULT 23.12.1899_15:00:00
TIMESTAMP DEFAULT '5/15/2006 7:00:00'
```

VARCHAR

Syntax

```
VARCHAR(n) [NOT NULL]
```

TurboDB Spaltentyp

String

Beschreibung

Wie CHAR

Beispiel

```
VARCHAR(40)
```

VARWCHAR

Syntax

```
VARWCHAR(n) [NOT NULL]
```

TurboDB Spaltentyp

WideString

Beschreibung

Wie WCHAR

Beispiel

```
VARWCHAR(20) NOT NULL
```

SMALLINT

Syntax

```
SMALLINT [NOT NULL]
```

TurboDB Spaltentyp

SmallInt

Beschreibung

Ganzzahl zwischen -32.768 und +32.767

Beispiel

```
SMALLINT
```

WCHAR

Syntax

```
WCHAR(n) [NOT NULL]
```

TurboDB Spaltentyp

WideString

Beschreibung

Unicode String bis N Zeichen. Da ein Unicode Zeichen 2 Bytes benötigt, ist die resultierende Feldgröße zwei mal die Anzahl der Zeichen. 1 <= N <= 32767

Beispiel

```
WCHAR(1000) DEFAULT '-'
```

5.4.5 Programmiersprache

Dieses Feature ist nur in TurboDB Managed verfügbar.

TurboSQL verfügt über Spracheelemente zur Erstellung von Routinen, die in SQL Befehlen verwendet werden können.

Benutzerdefinierte Funktionen

Funktionen können SQL Kommandos vereinfachen oder diese um zusätzliche Funktionalität erweitern. Sie werden entweder in TurboSQL geschrieben oder aus einer .NET Assembly importiert. Funktionen können über Input-Parameter verfügen und berechnen immer einen Rückgabewert. Sie dürfen keine Seiteneffekte haben. Funktionen werden mit den Statements CREATE FUNCTION und DROP FUNCTION verwaltet. Sie können für berechnete Indexe, berechnete Spalten und Gültigkeitsbedingungen verwendet werden.

Benutzerdefinierte Prozeduren

Prozeduren werden verwendet um komplexe Sequenzen von SQL Befehlen in einem einzigen Statement aufzurufen. Zum Beispiel können durch Verwendung einer Prozedur mehrere Datensätze verschiedener Tabellen in einem Schritt geändert werden. Prozeduren werden entweder in TurboSQL geschrieben oder aus einer .NET Assembly importiert. Prozeduren werden mit den Statements CREATE PROCEDURE und DROP PROCEDURE verwaltet.

Benutzerdefinierte Aggregate

Aggregate berechnen kumulierten Werte in Gruppen der Ergebnismenge. Sie können beispielsweise verwendet werden um das zweitgrößte Maximum, die Standardabweichung oder andere akkumulierte Werte aus einer gruppierten Ergebnismenge zu berechnen. Aggregate werden in einer .NET Assembly implementiert und mit den Statements CREATE AGGREGATE und DROP AGGREGATE verwaltet.

Statements

CREATE FUNCTION Statement
CREATE PROCEDURE Statement
CREATE AGGREGATE Statement
DROP FUNCTION/PROCEDURE/AGGREGATE Statement
DECLARE Statement
SET Statement
WHILE Statement
IF Statement
CALL Statement

Weiter Themen

Parameter mit .NET Assemblies austauschen

5.4.5.1 CALL Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
CALL procedure_name([argument, ...])
```

Beschreibung

Führt eine Stored Procedure mit den gegebenen Argumenten aus.

Beispiel

```
CALL LogLine('Das ist ein Test')
```

5.4.5.2 CREATE FUNCTION Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
CREATE FUNCTION function_name([:parameter_name data_type]...) RETURNS
data_type AS RETURN expression
```

```
CREATE FUNCTION function_name([:parameter_name data_type]...) RETURNS
data_type AS BEGIN statement [statement]... END
```

```
CREATE FUNCTION function_name([:parameter_name data_type]...) RETURNS
data_type AS EXTERNAL NAME [namespace_name.class_name].method_name,
assembly_name
```

Beschreibung

Eine Funktion kann überall da eingesetzt werden, wo ein Skalar erwartet wird, in Select Elementen, Suchbedingungen und anderen Ausdrücken.

namespace_name.class_name bezeichnet eine öffentliche (public) Klasse der Assembly.

method_name bezeichnet den Namen einer statischen public Funktion der Assembly. Die Funktion muss öffentlich (public) und nicht überladen (overloaded) sein . Die Parameter müssen mit denen der TurboSQL Funktion übereinstimmen (siehe "Parameter mit .NET Assemblies austauschen").

Beispiel

```
CREATE FUNCTION LastChar(:S WCHAR(1024)) RETURNS WCHAR(1) AS
RETURN SUBSTRING(:S FROM CHAR_LENGTH(:s) FOR 1)
```

```
CREATE FUNCTION Replicate(:S WCHAR(1024), :C INTEGER)RETURNS WCHAR(1024) AS
BEGIN
    DECLARE :I INTEGER
    DECLARE :R WCHAR(1024)
    SET :I = 0
    SET :R = ''
    WHILE :I < :C BEGIN
       SET :R = :R + :S
       SET :I = :I + 1
    END
    RETURN :R
END
```

```
CREATE FUNCTION CubicRoot(:X FLOAT) RETURNS FLOAT AS
EXTERNAL NAME [MathRoutines.TurboMath].CubicRoot,MathRoutines
```

5.4.5.3 CREATE PROCEDURE Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
CREATE PROCEDURE function_name([:parameter_name data_type]...) AS statement
```

```
CREATE PROCEDURE function_name([:parameter_name data_type]...) AS BEGIN
statement [statement]... END
```

```
CREATE PROCEDURE function_name([:parameter_name data_type]...) AS EXTERNAL
NAME [namespace_name.class_name].method_name,assembly_name
```

Beschreibung

namespace_name.class_name bezeichnet eine öffentliche (public) Klasse der Assembly.

method_name bezeichnet den Namen einer statischen public Funktion der Assembly. Die Funktion muss öffentlich (public) und nicht überladen (overloaded) sein . Die Parameter müssen mit denen der TurboSQL Funktion übereinstimmen (siehe "Parameter mit .NET Assemblies austauschen").

Beispiel

```
CREATE PROCEDURE Insert(:LastName WCHAR(40), :Salary INTEGER, :Department
WCHAR(40) AS BEGIN
    INSERT INTO Departments ([Name]) VALUES(:Department)
    INSERT INTO Employees (LastName, Salary, Department) VALUES(:LastName, :
Salary, :Department)
END
```

Das folgende Beispiel setzt die öffentliche, statische Methode *Send* der öffentlichen Klasse *SmtpClient* im Namensraum (namespace) *Email* der Assembly *Email.dll* voraus.Die Assembly muss im Verzeichnis der Datenbankdatei (tdbd) untergebracht sein.

```
CREATE PROCEDURE SendEmail(:Address WCHAR(100), :Subject WCHAR(100), :Content
WCHAR(100)) AS
EXTERNAL NAME [Email.SmtpClient].Send,Email
```

5.4.5.4 CREATE AGGREGATE Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
CREATE AGGREAGATE aggregate_name([:parameter_name data_type]...) RETURNS
data_type AS EXTERNAL NAME [namespace_name.class_name],assembly_name
```

Beschreibung

namespace_name.class_name bezeichnet eine öffentliche (public) Klasse der Assembly. Diese Klasse muss über einen Default Konstruktor verfügen und drei Methoden implementieren:

```
[C#]
public <class_name>()
public void Init()
public void Accumulate(<data_type>)
public <data_type> Terminate()
```

data_type muss mit den Paramentern einer TurboSQL Funktion übereinstimmen (siehe "Parameter mit .NET Assemblies austauschen").

Wenn ein benutzerdefiniertes Aggregat in einem SQL Statement verwendet wird, z.B: *SELECT MAX2(Salary) FROM Employees*, erzeugt die Engine eine Instanz der CLR Aggregat Klasse. Zu Beginn jeder Gruppe wird die *Init* Methode aufgerufen. Danach erfolgt für jeden Datensatz der Gruppe der Aufruf von *Accumulate* in der in der GROUP BY Klausel definierten Reihenfolge. Nachdem alle Datensätze der Gruppe akkumuliert sind, führt die Engine die Funktion *Terminate* aus, um das Ergebnis abzurufen. Jede angeforderte Ressource kann in *Terminate* freigegeben werden.

Beispiel

Der C# Code für ein Aggregat zur Berechnung des Maximums zweiter Ordnung sieht so aus:

```
namespace MathRoutines {

    public class SecondOrderMax {

        public SecondOrderMax() { }

        public void Init() {
            max = null;
            max2 = null;
        }

        public void Accumulate(double? x) {
            if (max == null || x > max) {
                if (max != null)
                    max2 = max;
                max = x;
            } else if (max2 == null || x > max2)
                max2 = x;
        }

        public double? Terminate() {
            return max2;
        }

        private double? max;
        private double? max2;
    }
}
```

Und so wird das Aggregat in TurboSQL importiert:

```
CREATE AGGREGATE Max2(:x DOUBLE) RETURNS DOUBLE AS
EXTERNAL NAME [MathRoutines.SecondOrderMax],MathRoutines
```

5.4.5.5 DROP FUNCTION/PROCEDURE/AGGREGATE Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

DROP FUNCTION | PROCEDURE | AGGREGATE

Beschreibung

Verwenden Sie DROP um die Funktion, Prozedur oder das Aggregat aus der Datenbank zu löschen.

5.4.5.6 DECLARE Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
DECLARE :variable_name data_type
```

Beispiel

```
DECLARE :i INTEGER
DECLARE :s CHAR(300)
```

5.4.5.7 IF Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
IF search_condition THEN if_statement [ELSE else_statement]
```

Beschreibung

Führt das Statement *if_statement* aus falls und nur falls die Bedingung *search_condition* wahr (True) ist . Falls *search_condition* nicht True liefert und *else_statement* gegeben ist, wird dieses Statement ausgeführt.

Beispiel

```
IF :s <> '' THEN SET :s = :s + ', '
IF :a > 0 THEN BEGIN
    SET :r = SQRT(:a)
END ELSE BEGIN
    SET :r = SQRT(-:a)
END
```

5.4.5.8 SET Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
SET :variable_name = expression
```

Beschreibung

Weist der Variable einen neuen Wert zu.

Beispiel

```
SET :LastName = 'Miller'
```

5.4.5.9 WHILE Statement

Dieses Feature ist nur in TurboDB Managed verfügbar.

Syntax

```
WHILE search_condition statement
```

Beschreibung

Führt *statement* aus, so lange *search_condition* wahr (True) ist.

Beispiel

```
DECLARE :I INTEGER
WHILE :I < 100 SET :I = :I + 1
DECLARE :I INTEGER
WHILE :I < 100 BEGIN
    SET :I = :I + 1
END
```

5.4.5.10 Parameter mit .NET Assemblies austauschen

Dieses Feature ist nur in TurboDB Managed verfügbar.

Assemblies, die externe Funktionen, Prozeduren oder Aggregate zur Verfügung stellen, müssen im Verzeichnis der Datenbankdatei untergebracht sein.

Beim Aufruf von Methoden aus .NET Assemblies werden die Paramter von TurboSQL nach CLR gemappt, gemäß der nachfolgenden Tabelle. Wie zu sehen ist, besteht ein Unterschied zwischen Funktionen auf der einen und Prozeduren und Aggregaten auf der anderen Seite.

Funktionen haben nur Input-Parameter und es wird NULL zurückgegeben, falls eines der Argumente NULL ist. In diesem Fall wird die CLR Methode überhaupt nicht ausgeführt. Aus diesem Grund wird NULL nie an eine benutzerdefinierte CLR Funktion übergeben und die CLR Methode darf nur mit CLR-Typen deklariert werden, die keine NULL-Werte kennen (non-nullable). Zum Beispiel:

```
CREATE FUNCTION Log2(:x DOUBLE NOT NULL) RETURNS DOUBLE NOT NULL AS EXTERNAL
NAME [MyNamespace.MyClass].MyMethod,MyAssembly
```

korrespondiert mit dieser Definition in C#:

```
public class MyClass {
    static public double MyMethod(double x) {...}
}
```

Weil das Argument nicht zwischen 0 und NULL unterscheidet, kann *Log2* nie mit NULL-Werten verwendet werden. Sieht die Deklaration dagegen so aus

```
CREATE FUNCTION Log2(:x DOUBLE) RETURNS DOUBLE AS EXTERNAL NAME [MyNamespace.
MyClass].MyMethod,MyAssembly
```

ist obige Definition gültig in C# . Wird Log2 mit einem NULL-Argument aufgerufen, liefert die Engine NULL ohne die CLR Methode aufzurufen.

Diese Regel gilt nur für Funktionen; CLR Prozeduren und Aggregate werden immer aufgerufen, auch für Argumente die NULL sind. Daher müssen die Parameter und der Rückgabetyp der CLR Definition in der Lage sein, NULL-Werte zu transportieren. TurboSQL tut das indem *null* (*Nothing* in Visual Basic) übergeben wird, was für *CHAR*- und *LONGVARBINARY*-Datentypen offensichtlich ist. Um auch mit anderen Datentypen wie *Int64* oder *DateTime* in dieser Weise verfahren zu können, muss der Nullable-Wrapper verwendet werden.

```
CREATE PROCEDURE TestProc(:x DOUBLE) AS EXTERNAL NAME [MyNamespace.MyClass].
MyMethod,MyAssembly
```

wird daher abgebildet durch

```
public class MyClass {
    static public void MyMethod(Nullable<double> x) {...}
}
```

oder

```
public class MyClass {
    static public void MyMethod(double? x) {...}
}
```

oder in Visual Basic:

```
Public Class MyClass
    Public Shared Sub MyMethod(X As Nullable(Of Double))
      ...
    End Sub
End Class
```

Weitere Informationen zu Nullable-Wrapper sind in der MSDN zu finden, als Suchbegriff kann *Nullable class* verwendet werden.

TurboSQL Typ	Non-Nullable CLR Typ	Nullable CLR Typ
BYTE	*System.Int64*	*Nullable<System.Int64>*
SMALLINT	*System.Int64*	*Nullable<System.Int64>*
INTEGER	*System.Int64*	*Nullable<System.Int64>*
BIGINT	*System.Int64*	*Nullable<System.Int64>*
FLOAT	*System.Double*	*Nullable<System.Double>*
AUTOINC	*System.Int64*	*Nullable<System.Int64>*
BOOLEAN	*System.Int64 (0 corresponds to false, all other values to true)*	*Nullable<System.Int64>*
ENUM	*System.Int64*	*Nullable<System.Int64>*
GUID	*System.Guid*	*Nullable<System.Guid>*
LINK	*System.Int64*	*Nullable<System.Int64>*
RELATION	*System.Int64[]*	*System.Int64[]*
CHAR(X), VARCHAR (X)	*System.String*	*System.String*
WCHAR(X), VARWCHAR(X)	*System.String*	*System.String*
LONGVARCHAR	*System.String*	*System.String*
LONGVARWCHAR	*System.String*	*System.String*
LONGVARBINARY	*System.Byte[]*	*System.Byte[]*
TIME	*System.DateTime*	*Nullable<System.DateTime>*
DATE	*System.DateTime*	*Nullable<System.DateTime>*
TIMESTAMP	*System.DateTime*	*Nullable<System.DateTime>*

5.5 dataWeb Produkte und Werkzeuge

Es gibt verschiedene Werkzeuge, die Ihnen die Arbeit mit TurboDB erleichtern.

TurboDB Viewer: Visuelles Werkzeug zur Verwaltung und Bearbeitung von TurboDB Datenbanken, Tabellen und Indexen.

TurboDB Pilot: SQL Konsole zur Verwaltung und Bearbeitung von Single-File Datenbanken ab Tabellen Level 5.

dataWeb Compound File Explorer: Visuelles Verwaltungswerkzeug für TurboDB Single-File Datenbanken

TurboDB Workbench: Konsolenprogramm zum Bearbeiten von TurboDB Tabellen und Indexen.

TurboDB Data Exchange: Konsolenprogramm zum Importieren und Exportieren von Daten in und aus TurboDB Tabellen.

TurboDB Server: Datenbankserver für TurboDB-Tabellen: Erlaubt bis zu 100 gleichzeitige Sessions auf eine Datenbank (bald verfügbar).

TurboDB Studio: RAD Werkzeug für die Entwicklung von Windows Anwendungen mit TurboDB Engine.

Weitere Information zu TurboDB Produkten und Tools finden Sie auf der dataweb Homepage www.dataweb.de.

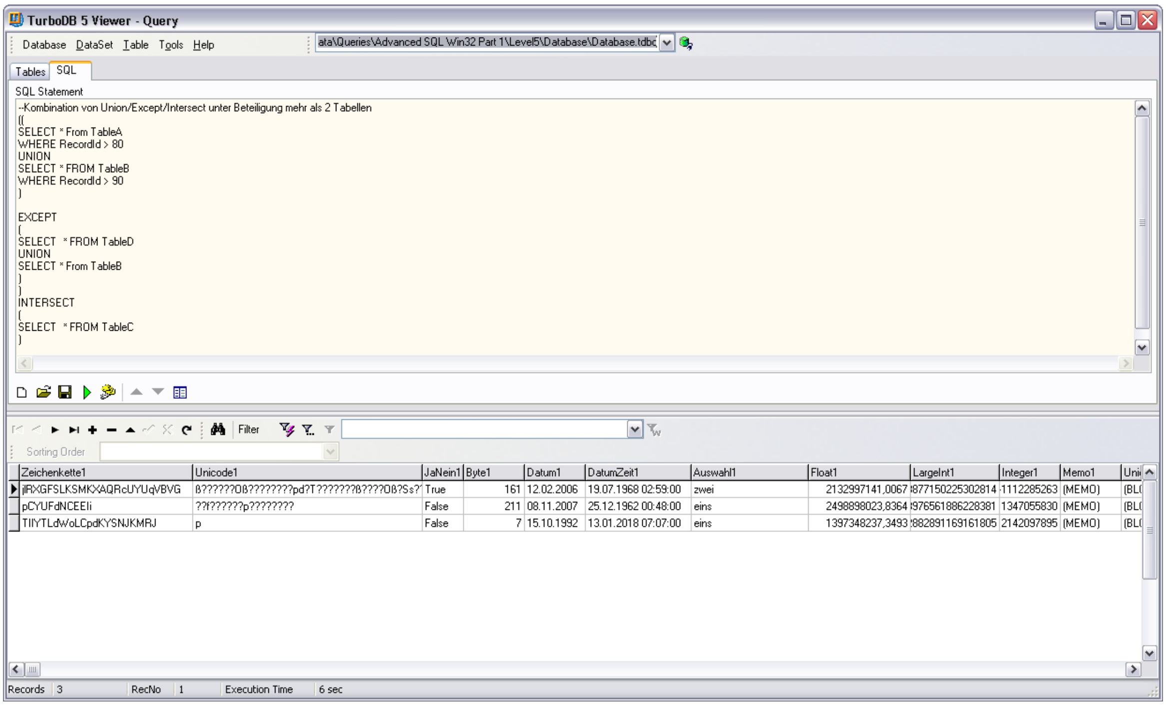
5.5.1 TurboDB Viewer

TurboDB Viewer ist ein visuelles Werkzeug um Tabellen und Indexe zu verwalten und um Tabellen zu editieren. Das können Sie mit dem Programm machen:

- Tabelleninhalte betrachten und editieren
- SQL Abfragen ausführen

- Datenbanktabellen entwerfen
- Datenbanktabellen ändern
- Indexe erstellen. ändern oder löschen
- Volltexindexe erstellen. ändern oder löschen

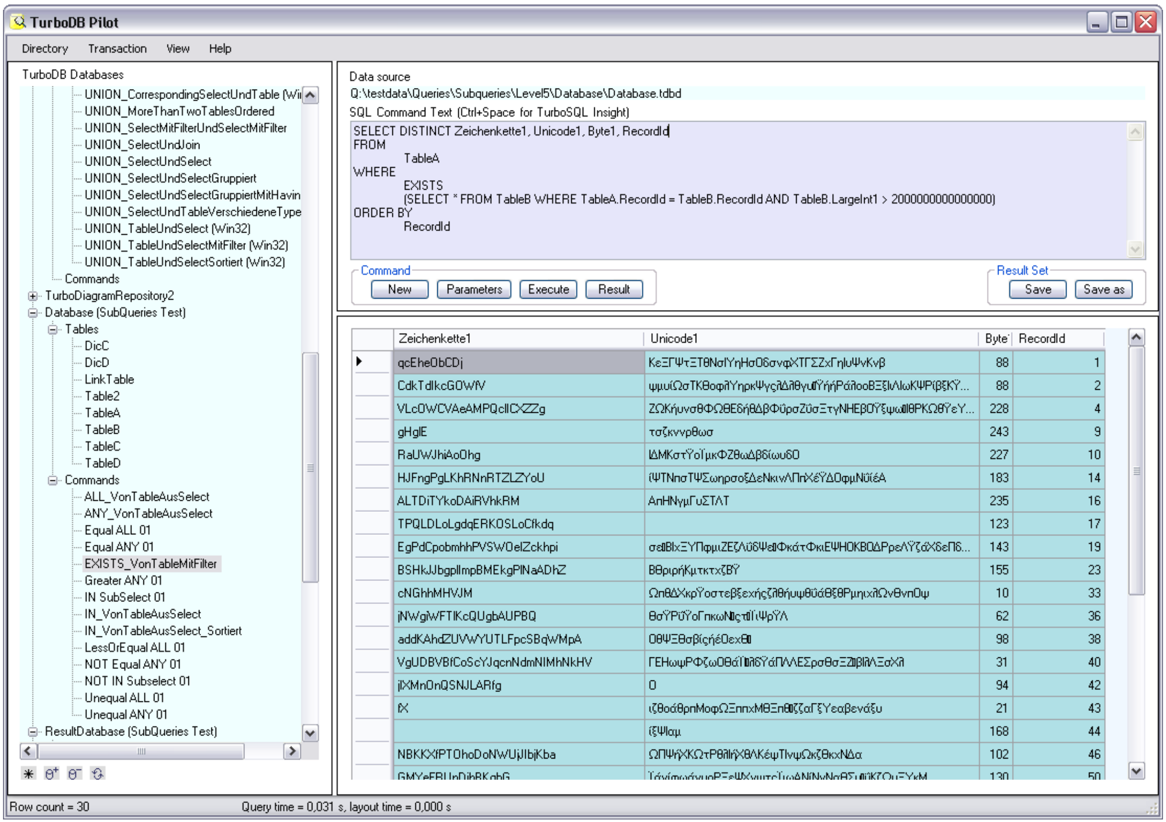
TurboDB Viewer ist im Lieferumfang von TurboDB enthalten und unterstützt alle Tabellen Level und Features. Sie finden ihn entweder im Startmenü oder als ausführbare Datei namens *TurboDB5Viewer.exe*.



Beim Erstellen von Datenbanken für TurboDB Managed muss beachtet werden, dass nur Features verwendet werden, die TurboDB Managed unterstützt werden. Daher ist für TurboDB Managed der Einsatz von TurboDB Pilot empfehlenswert, der selbst auf der managed Datenbank Engine basiert.

5.5.2 TurboDB Pilot

TurboDB Pilot ist eine SQL Konsole für TurboDB Level 5 Datenbanken, der auf der Managed Database Engine basiert.



5.5.3 dataWeb Compound File Explorer

Dies ist das Management Werkzeug zu dataWeb Compound File-Technologie. Da eine TurboDB Datenbank-Datei ein Compound File ist, kann mit diesem Werkzeug der Inhalt einer Datenbank-Datei angesehen und editiert werden. Es wird mit allen TurboDB Produkten ausgeliefert und ist im Start-Menü als Compound File Explorer zu finden. Die zugehörige ausführbare Datei heißt CfExplorer.exe.

Im Einzelnen hat der Compound File Explorer folgende Möglichkeiten:

- TurboDB Single-File Datenbank-Datei (*.tdbd) öffnen und die enthaltenen Einzeldateien ansehen.
- Einzeldatei löschen oder umbenennen
- Einzeldatei öffnen
- Einzeldateien aus dem Compound File extrahieren oder in das Compound File aufnehmen.

Damit kann man den Compound File Explorer auch benutzen, um aus einer Verzeichnis-Datenbank eine Datenbank-Datei zu machen.

5.5.4 TurboDB Workbench

tdbwkb ist ein kleines textbasiertes Freeware Tool zur Verwaltung von TurboDB Tabellen. Es ist für Windows erhältlich und kann auch von unserer Website http://www.turbodb.de heruntergeladen werden

- Tabellen erzeugen, ändern, umbenennen, löschen
- Tabelleninhalte betrachten

- Tabellenstruktur ansehen und Tabelle reparieren
- Indexe für Tabellen erstellen, ändern und löschen

Beim Start des Programms wird das *tdbwkb* Prompt angezeigt, hier können Sie die verschiedenen Kommandos eingeben. Geben Sie help ein um eine Liste mit den möglichen Befehlen anzuzeigen.

Hier beispielhaft eine *tdbwkb* Session um die verfügbaren Features zu demonstrieren.

```
c:\programme\dataweb\TurboDBStudio\bin\tdbwkb
dataWeb Turbo Database Workbench Version 4.0.1 (TDB 6.1.6)
Copyright (c) 2002-2005 dataWeb GmbH, Aicha, Germany
Homepage http://www.dataWeb.de, Mail dataWeb Team
Type 'help' to get a list of available commands.

tdbwkb> help
  Abbreviations are not allowed. The commands are:
  altertable    Modifies an existing table.
  bye           Ends the tdbwkb session.
  cd            Changes the current directory.
  debug         Toggles debug mode. (Debug mode prints log messages.)
  delindex      Deletes an index from a table.
  deltable      Deletes all files of a table.
  help          Prints this list of commands.
  newftindex    Create a new full text index for a table.
  newindex      Creates a new index for a table.
  newtable      Creates a new table.
  pwd           Prints current working directory.
  show          Shows a rough preview of the table.
  switchdb      Opens another database.
  rename        Renames a table.
  repair        Rebuilds a table and all its indexes.
  tableinfo     Shows the description of a table.
  Type help <cmd> to get more specific help for a command.
  Note: You may also use tdbwkb in batch mode by appending the command
directly
  to the call. Example:
  tdbwkb tableinfo mytable

tdbwkb> newtable animals
S40Name,A'Land,Water,Air'Area,PImage,MDescription,N'Name'RecordId
Creating table animals.dat with these columns:
  1 S40  Name
  2 A    Area, Values = Land,Water,Air
  3 P    Image
  4 M    Description
  5 N    RecordId
tdbwkb> tableinfo animals
Retrieving structure of table animals.dat...
Table columns:
  1 S40  Name
  2 A    Area, Values = Land,Water,Air
  3 P    Image
  4 M    Description
  5 N    RecordId
Indexes:
animals.inr                   RecordId:4
animals.id                    Name:40

tdbwkb> altertable animals n2=S40Family
Restructuring table animals.dat to these columns:
  1 S40  Name
  2 S40  Family
  3 A    Area, Values = Land,Water,Air
  4 P    Image
  5 M    Description
  6 N    RecordId

tdbwkb> newindex animals byfamily Family,Name

tdbwkb> tableinfo animals
Retrieving structure of table animals.dat...
```

```
Table columns:
  1 S40  Name
  2 S40  Family
  3 A    Area, Values = Land,Water,Air
  4 P    Image
  5 M    Description
  6 N    RecordId
Indexes:
animals.inr                  RecordId:4
animals.id                   Name:40
byfamily.ind                 Family:40, Name:40
```

5.5.5 TurboDB Studio

Das ist unser Werkzeug um Turbo Datenbank-Anwendungen unter Windows zu erzeugen. Mit TurboDB Studio erzeugen Sie Formulare und Berichte für interaktive Programme und Ausdrucke.

Sie erstellen in wenigen Stunden Ihre eigenen Anwendungen (Exe) mit eigenem Namen, Logo, Menü und Werkzeugleiste. TurboDB Entwickler könnnen mit TurboDB Studio Ihre TurboDB Datenbank auf einfachste Weise verwalten, Daten eingeben und Berichte drucken.

Mit TurboDB Studio können Sie

- TurboDB Datenbanken komfortabler verwalten als mit TurboDB Viewer
- Prototypen für TurboDB Anwendungen sehr schnell erstellen
- Ihren Kunden ein zusätzliches Werkzeug zur Verfügung stellen (einschließlich Berichtwesen)
- Vollständige Windows Anwendungen basierend auf TurboDB erstellen

Der Vorgänger von TurboDB Studio hieß Visual Data Publisher. Weitere Informationen zu TurboDB Studio finden Sie unter http://www.dataweb.de/de/vdp/index.html.

5.5.6 TurboDB Data Exchange

Ein konsolenbasiertes Freeware Tool, das Daten von und in TurboDB Tabellen lesen und schreiben kann. Es wird mit allen TurboDB-Produkten mitgeliefert und liegt als TdbDataX.exe im Installationsverzeichnis. Sie können die aktuelle Version jederzeit von der dataWeb Homepage herunterladen.

Von TurboDB Data Exchange unterstützte Formate sind Text, dBase, TurboDB, XML und ADO.

6 Nachschlagen

6.1 Handbuch

Die 584 Seiten starke, gedruckte Variante dieser Dokumentation ist bei Books on Demand unter der ISBN 3833451068 erschienen und kann in jedem Buchladen bestellt werden.

Alternativ dazu bieten wir eine immer auf dem aktuellen Stand gehaltene Version als PDF im A4 Format zum Downloaden und Drucken:
http://www.dataweb.de/de/support/dokumentation/turbodbstudio/TurboDBStudio.pdf

6.2 Tastenkürzel

6.2.1 Tastenkürzel

Mit den folgenden Tastenkombinationen können Sie TurboDB Studio effektiv über die Tatstatur bedienen, ohne die Maus zu benutzen.

Allgemeine Tastenkürzel
Tastenkürzel im Datenfenster
Tastenkürzel im Texteditor

6.2.2 Allgemeine Tastenkürzel

Taste	Bedeutung
Strg + F4	Schließen des aktiven Unterfensters
Alt + F4	Beenden von TurboDB Studio
Umschalt + F9	Drucken des aktuellen Projektelements
Strg + F9	Starten des aktuellen Projektelements
Alt + F9	Entwerfen des aktuellen Projektelemts
F10	Menüzeile aktivieren
Alt + F10	Lokales Menü öffnen
F11	Projektfenster ein- und ausschalten
Strg + F11	Zwischen Projekt- und Dokumentenfenstern wechseln
Alt + F11	Tabellenstrukturfenster anzeigen
Entf	Markierte Textstelle, aktuelles Bild, Projektelement oder selektierte Maskenelemente löschen
Strg + X	Markierte Textstelle, aktuelles Bild, Projektelement oder selektierte Maskenelemente in die Zwischenablage verschieben
Strg + V	Markierte Textstelle, aktuelles Bild, Projektelement oder selektierte Maskenelement aus der Zwischenablage einfügen
Strg + C	Markierte Textstelle, aktuelles Bild, Projektelement oder selektierte Maskenelemente in die Zwischenablage kopieren
Strg + Z	Letzte Aktion zurücknehmen

6.2.3 Tastenkürzel im Datenfenster

Taste	Bedeutung
F1	Kontextsensitive Hilfe zur aktuellen Situation
Umschalt + F1	Online-Hilfe beim Inhaltsverzeichnis öffnen
F3	Wiederholen der letzten Suche
Umschalt + F3	Direkte Suche über Index
Strg + F3	Volltextsuche
F4	Öffnen der Auswahlliste, Nachschlagetabelle oder angekoppelten Tabelle
F5	Zum nächsten markierten Datensatz gehen
Umschalt + F5	Zum vorherigen markierten Datensatz gehen
F7	Umschalten zwischen Formularsicht und Tabellensicht
Umschalt + F7	Index für Sortierung auswählen
Strg + F7	Nur markierte Datensätze anzeigen
Alt + F7	Alle Datensätze anzeigen
F8	Umschalten der Markierung für den aktuellen Datensatz
Umschalt + F8	Alle Markierungen entfernen
Strg + Ä	Editiermodus für Datensätze ein- und ausschalten
Strg + F	Suchen mit Bedinung
Strg + N	Im Datenfenster Modus für Neueingabe ein- und ausschalten
Strg + O	Makro ausführen
Strg + R	Suchen und Ersetzen in Datensätze
Strg + W	Wählverbindung aufbauen

6.2.4 Tastenkürzel im Texteditor

Taste	Bedeutung
Strg + F1	Hilfe zum Kommando oder zur Funktion an der Kursorposition (Nur im Modul- und Datenbankjob-Editor)
F3	Wiederholen der letzten Suche
Strg + F	Suchen nach Textstelle
Strg + R	Suchen und Ersetzen von Textstellen

6.3 Glossar

Dieses Kapitel enthält alphabetisch geordnet Erklärungen zu den wichtigsten Begriffen im Zusammenhang mit Datenbanken und *TurboDB Studio*.

6.3.1 ADL-System

ADL steht für Automatic Data Link und bezeichnet ein Verfahren der Kopplung von Tabellen über automatisch vergebene Nummern, sog. Autonummern. Um zwei Tabellen per ADL zu koppeln, benötigt die eine ein Auto-Nummer-Feld und einen ID-Index, die andere ein Koppel- oder L-Feld zur Aufnahme der Autonummer des angekoppelten Datensatzes. Auf diese Weise werden jeweils mehrere Datensätze der einen Tabelle mit einem Datensatz der anderen Tabelle verknüpft (1-n Beziehung).

Relations- oder R-Felder dagegen bauen Verknüpfungen von mehreren Datensätzen der einen zu mehreren Datensätzen der anderen Tabelle auf. Dazu richtet TurboDB Studio intern eine dritte Tabelle ein, die mittels zweier Koppelfelder diese Verknüpfung herstellt (n-m Beziehung).

Über ADL verknüpfte Tabellen sind sehr einfach zu warten und äußerst schnell bei Abfragen und Berichten. Vor allem aber unterstützen Koppel- und Relationsfelder die Eingabe von Verknüpfungen und erleichtern damit das Pflegen der Datenbank erheblich.

6.3.2 Applikationsmodul

Ein Modul, das nicht an ein Formular gebunden ist, sondern von überall in der Applikation aufgerufen werden kann. Applikationsmodule sind im Projektfenster als Knoten unterhalb einer Familie eingetragen. Die zweite Art von Modulen in TurboDB Studio sind die Formularmodule.

6.3.3 Ausgabeformat

Mit einem Ausgabeformat legen Sie fest, wie Ihre Datensätze Datenbankjobs und im Text-Anzeigeelement eines Berichtes dargestellt werden. Ein Beispiel für ein einfaches Ausgabeformat mit drei Spalten der Breite 20 Zeichen ist z.B. *$Name:20 $Vorname:20 $Abteilung:20*

Siehe auch: "Ausgabeformate".

6.3.4 Autonummer

Jede Tabelle kann ein Spalte enthalten, die eine eindeutige automatisch vergebene Nummer enthält. Diese Nummer wird als Auto-Nummer bezeichnet und spielt besonders im Zusammenhang mit dem ADL-System eine wichtige Rolle. Dort werden Datensätze einer Tabelle an Datensätze einer anderen Tabelle angekoppelt, indem im Koppelfeld der Primärtabelle die Auto-Nummer des angekoppelten Datensatzes abgespeichert wird. Das ADL-System sorgt dafür, dass im verknüpften Datensatz anstelle der Auto-Nummer, die wesentlich aussagekräftigere Index-Information des ID-Index angezeigt wird.

6.3.5 Bedingung

Eine Bedingung ist ein Satz der entweder wahr oder falsch sein kann. Z.B. 'Das Auto ist rot' oder auf Datenbanken bezogen 'Name ist Müller'. Bedingungen werden verwendet, um nach Datensätzen zu suchen (nämlich nach denen, auf die die Bedingung zutrifft) oder um festzulegen, welche Datensätze ausgedruckt werden sollen. Dazu müssen die Bedingungen allerdings in einer Form angegeben werden, die TurboDB Studio versteht, im oberen Beispiel lautet sie dann 'Name ist "Müller"'. In dieser Datenbank-gemäßen Form werden Bedingungen dann auch Selektionen genannt.

6.3.6 Bericht

Ein Bericht dient zum Drucken von Berichten, Etiketten oder anderen Auswertungen Ihrer Datenbestände. Berichte werden im Berichteditor entworfen, der ähnlich funktioniert wie der Formulareditor. Berichte können Sie also sehr schnell und einfach erstellen, die einzelnen Felder sind mit der Maus millimetergenau positionierbar, und Sie haben die Möglichkeit, verschiedene Schriftarten und -größen zu verwenden sowie Bilder auszudrucken.

6.3.7 Datenbankjob

Datenbankjobs benötigen Sie für komplexe Reports über Ihre Datensätze oder zum Erstellen von Serienbriefen. Das Aussehen des Reports oder Serienbriefen legen Sie in einer Text-Datei fest, die Sie mit dem Texteditor bearbeiten können. Ein Datenbankjob kann auch Meldungen ausgeben oder Benutzereingaben in einer Dialogbox abfragen.

Siehe auch: Datenbankjobs erstellen

6.3.8 easy

Ehemaliger Name für die Programmiersprache von *TurboDB Studio*. Die aktuelle Bezeichnung ist *TurboPL*.

6.3.9 Einheit, logische

Die logische Einheit wird in Datenbankjobs für alle Maßangaben benutzt. Wie groß eine logische Einheit tatsächlich ist, legt der Steuerbefehl MM fest. Mit MM 10 zum Beispiel entspricht eine logische Einheit 1 mm.

6.3.10 Formel

Eine Formel ist eine Rechenvorschrift, die bestimmt, wie aus gegebenen Werten ein neuer Wert berechnet wird. Z.B. '3+4' oder 'sin(5.3*8.3245)'. Die Formeln, die *TurboDB Studio* versteht, werden als Ausdrücke bezeichnet und können sehr viel mehr als einfach nur Zahlen ausrechnen. Z.B. berechnet der Ausdruck 'Length(Name)' die Anzahl der Buchstaben im Datenfeld Name oder 'heute - 15.10.1994' die Anzahl der Tage die seit dem 15.10.1994 vergangen sind.

6.3.11 Formular

In Formularen werden Datensätze neu eingegeben und editiert. Um ein Formular zu gestalten, benutzen Sie den Formulareditor, wo Sie die einzelnen Felder mit der Maus positionieren und alle Eigenschaften einstellen können. Ein Formular kann mehrere Seiten umfassen, über Eingabekontrollen verfügen, sowie Bilder und Makros enthalten.

6.3.12 formularbezogen

Ein Makro ist Formular-bezogen, wenn es sich auf ein Formular bezieht statt auf die Applikation.

Siehe auch

Applikations-Module und Formular-Module

6.3.13 Formularmodul

Das Formularmodul ist ein Modul, welches einem Formular zugeordnet ist. Dadurch kann das Programm im Formularmodul auf die speziellen Eigenschaften des Formulars, wie zum Beispiel seine Steuerelemente zugreifen. Im Projektfenster ist das Formularmodul ein Knoten unterhalb des Formulars zu dem es gehört.

6.3.14 IDIndex

Der ID-Index (Identifikationsindex) oder Standard-Index ist ein spezieller Index, der bereits beim Entwurf der Tabellenstruktur, genauer bei der Definition des Auto-Nummern-Feldes, festgelegt wird.

Als Standard-Index wollen wir einen solchen Index bezeichnen, der den natürlichen Zugriff zu den Daten erlaubt. In einer Adressdatei wird man dazu wahrscheinlich eine Kombination aus Name und Vorname verwenden oder in einer Firmendatei den Firmennamen. In einer Artikeldatei entweder Artikelnummer oder Bezeichnung. Seine besondere Bedeutung erhält dieser Index erst im Zusammenhang mit dem ADL-System.

Aber auch in einzelnen Datendateien kann es nie schaden, einen ID-Index anzulegen, da Sie dadurch ein Standardsortierkriterium erhalten, und außerdem ist eine Datei mit ID-Index für nachträgliches Verkoppeln bereits vorbereitet.

6.3.15 Index

Ein Index ist so etwas wie ein Inhaltsverzeichnis Ihrer Tabelle und erlaubt einen schnellen Zugriff auf einen bestimmten Datensatz. Wenn Sie z.B. nach einem bestimmten Nachnamen suchen, kann das in einer großen Datei ziemlich lange dauern. Haben Sie aber einen nach dem Nachnamen sortierten Index angelegt, passiert die Suche in Sekundenbruchteilen.

Siehe auch: Einen Index erstellen

6.3.16 Indexbeschreibung

Eine Indexbeschreibung definiert eindeutig die Zusammensetzung eines Indexes. In folgenden Situationen werden Indexbeschreibungen benötigt:

- ID-Index-Definition
- Statische Links
- Sortierkriterium

Eine Indexbeschreibung darf grundsätzlich nur nicht-relationale Tabellenzugriffe enthalten. Bei Hierarchischen Indexbeschreibungen sind bis zu zehn Hierarchiestufen möglich. Der Ausdruck einer Indexformel darf maximal 40 Zeichen umfassen.

Beispiele für legale Indexbeschreibungen:

```
Name
Name, Vorname
LeftStr(Name, 5), LeftStr(Vorname, 3)
PLZ + " " + Ort
```

6.3.17 Job

Kurzform von Datenbankjob.

6.3.18 Koppelfeld

Das Koppelfeld, auch L-Feld genannt, nimmt die Autonummer eines anderen Datensatzes auf und koppelt ihn dadurch an den Datensatz, der das Koppelfeld enthält. Da die Anzeige der Autonummer dem Benutzer nicht viel nützen würde, erscheint stattdessen die im ID-Index gespeicherte Information. Dieses Verhalten wird als natürlicher Zugriff bezeichnet.

Siehe auch

ADL-System

6.3.19 Koppelfeld-Notation

Handelt es sich bei einer Tabellenspalte um ein Koppelfeld, so kann daran, durch einen Punkt getrennt, ein Spaltenname der angekoppelten Tabelle angefügt werden. Diese Vorgehensweise ist kaskadierbar.

Die Besonderheit dieses Zugriffs liegt darin, dass TurboDB Studio bei jeder Auswertung den zugehörigen Datensatz in der angekoppelten Tabelle liest. Damit wird die Verfügbarkeit der Daten auch im Editier-Modus etc. sichergestellt. Deshalb sind solche Zugriffe besonders in Formelfeldern im Formular und Bericht empfehlenswert.

Zwingend erforderlich ist der Zugriff über Koppelfeld-Notation wenn mehrere Verknüpfungen zwischen zwei Tabellen vorhanden sind.

Beispiel

Erste Tabelle: ORTE
Spalten: Land, PLZ, Ort, Einwohnerzahl, Vorwahl
Zweite Tabelle: ADRESSEN
Spalten: Name, Vorname, Straße, Stadt(Koppelfeld auf ORTE)

Um nun in einem Formular der Tabelle ADRESSEN die Informationen des angekoppelten Datensatzes der ORTE-Tabelle anzuzeigen, können Formelfelder mit folgenden Ausdrücken zum Einsatz kommen:

```
Stadt.Land
Stadt.PLZ
Stadt.Ort
```

6.3.20 Label

Die Namen von Tabellenspalten werden vor allem in der Turbo Datenbank für DOS als Label bezeichnet. Diese Labels sind in der Turbo Datenbank für DOS gleichzeitig die Beschriftungen für die Eingabefelder in der Maske. Unter TurboDB Studio sind die Beschriftungen völlig unabhängig von den Spaltennamen. Aus diesem Grund wird der Begriff Label hier nicht mehr benutzt.

6.3.21 Laufzeitfehler

Ein Fehler, der während der Ausführung eines TurboPL-Makros auftritt und zum Abbruch führt. Laufzeitfehler können mit try/except abgefangen und behandelt werden. Die alte Methode mit EC 1 wird nicht mehr empfohlen.

Siehe auch: Makro-Fehler behandeln

6.3.22 Liste

Listen werden seit Visual Data Publisher 2 nicht mehr unterstützt.

Um bestehende Listen in Datenbankjobs zu konvertieren steht das Programm LIST2JOB zur Verfügung, das Sie im BIN-Verzeichnis Ihrer VDP finden. LIST2JOB konvertiert auch Listen der Turbo Datenbank für DOS.

6.3.23 Maske

Unter dem Begriff Maske werden die beiden recht ähnlichen Projektelemente Formular und Berichtzusammengefaßt.

6.3.24 Modul

Ein Modul ist ein Textdatei, die Prozeduren enthält. Die Prozeduren können als Makro aufgerufen werden.

6.3.25 modal

Unter Windows wird ein Fenster als modal bezeichnet, wenn der Benutzer nur in dieses Fenster eingeben kann, während das Fenster angezeigt ist. Die anderen Fenster des gleichen Programms sind solange still gelegt. In TurboDB Studio sind z.B. alle Dialogfenster modal. Darüber hinaus verwendet TurboDB Studio für einige spezielle Aufgaben modale Datenfenster. Z.B. bei der Kopplung von Datensätzen an ein Koppel- oder Relationsfeld oder für die Anzeige der Nachschlagetabelle. Solange das modale Datenfenster angezeigt ist, kann kein anderes Fenster des TurboDB Studio aktiviert werden. Im Gegensatz zu Dialogfenstern erlaubt das modale Datenfenster jedoch den Zugriff auf die Menüleiste.

6.3.26 Nachschlagetabelle

Eine Nachschlagetabelle dient dazu, dass Werte einer Tabelle als Auswahlmöglichkeiten für die Eingabe in eine andere Tabelle angeboten werden. Nachschlagetabellen werden im Formulareditor in den Eigenschaften einer *ComboBox* unter *Werteliste* und *Wertelistentyp* definiert.

6.3.27 NatürlicherZugriff

Das ADL-System ermöglicht die Kopplung von Datensätzen über automatisch vergebenen Nummern, sog. Auto-Nummern. Diese Nummer wird im Koppelfeld des übergeordneten Datensatzes gespeichert. Die Anzeige dieser Nummer, würde dem Benutzer allerdings nicht viel sagen. Aus diesem Grund wird statt dessen die Indexinformation des ID-Index ausgegeben. Den Aufbau dieses Index ist frei wählbar. In einer Kunden-Tabelle würde man also statt einer nichtssagenden Nummer Nach- und Vorname angezeigen.

6.3.28 Primärtabelle

Die Haupttabelle für ein Formular, Datenbankjob oder Bericht wird Primärtabelle genannt. Auf Felder der Primärtabelle kann man gefahrlos ohne vorangestellten Tabellennamen verweisen. In Berichten und Datenbankjobs werden in erster Linie die Datensätze der Primärtabelle abgearbeitet, alle anderen Tabellen werden in Beziehung zu dieser Tabelle betrachtet.

6.3.29 Projekt

TurboDB Studio benutzt Projekte, um Tabellen, Formulare, Berichte, Datenbankjobs und Module zusammenzufassen. Das Projekt enthält die Liste der Dateipfade und Namen aller beteiligten Elemente sowie weitere Informationen wie Zugriffsrechte, Statische Verknüpfungen, Spaltenformate und Makros.

Weitere Informationen finden Sie im Kapitel "Projekte verwalten".

6.3.30 Relationales Datenbankprogramm

Ein relationales Datenbankprogramm organisiert die gespeicherten Informationen in einer oder in mehreren Tabellen.

6.3.31 Relationsfeld, Definition

Während das Koppel- oder L-Feld mehrere Datensätze an einen Datensatz ankoppelt, können die angekoppelten Datensätze bei einem Relationenfeld mit mehreren übergeordneten Datensätzen verbunden sein. Typisches Beispiel: Ein Autor schreibt viele Bücher, aber ein Buch kann auch mehrere Autoren haben. Intern wird diese M-zu-N-Verknüpfung über eine zusätzliche Tabelle realisiert, die mit zwei Koppelfeldern jeweils auf die beiden Partner einer Verknüpfung zeigt.

Siehe auch: ADL-System

6.3.32 Rollback

Wenn eine Transaktion nicht komplett korrekt durchgeführt werden kann, werden alle Änderungen dieser Transaktion zurückgenommen. Diesen Vorgang bezeichnet man als Rollback.

Siehe auch: Transaktion

6.3.33 Selektion

Eine Selektion ist eine als Text formulierte Bedingung für die Auswahl von Datensätzen, wie z.B. Name ist "Meier". Selektionen werden bei der Suche mit Bedingung verwendet, zur Eingabeüberprüfung in Formularen und in Berichten und Datenbankjobs zur Auswahl der auszudruckenden Datensätze.

6.3.34 Serienbrief

Ein Serienbrief ist eine Mitteilung, die an mehrere Empfänger einer Adressdatenbank verschickt wird. Ein Serienbrief kann in TurboDB Studio entweder als Bericht oder als Datenbankjob realisiert werden. Bei einem Datenbankjob verwenden Sie das Kommando letter statt report zu Beginn, um einen Serienbrief zu definieren.

Siehe auch: "Datenbankjobs".

6.3.35 Sortierung

Die Sortierung oder Sortierordnung bestimmt die Reihenfolge, in der Datensätze angezeigt oder ausgedruckt werden. In den meisten Fällen wird sie dadurch festgelegt, dass man einen Index angibt, der die gewünschte Sortierung besitzt.

Siehe auch: Zugriff

6.3.36 Speicher-Datei

Mit dem speziellen Dateinamen RAMTEXT kann mit den bekannten Funktionen zum Öffnen von Dateien (Reset, Rewrite oder TAppend) eine Speicher-Datei angelegt werden. Diese Speicher-Datei wird mit den üblichen Funktionen (Read, Write, ...) weiterbearbeitet wie eine normale Datei. Da sie sich jedoch vollständig im Hauptspeicher befindet, gehen alle Schreib- und Leseoperationen sehr schnell vonstatten. Hier ein Beispiel für die Verwendung der Speicher-Datei zum Eintragen eines Textes in ein Memo-Feld:

```
PROCEDURE READ_TEXT
    VarDef nFI : REAL;
    .. Speicher-Datei wird angelegt und geöffnet
    nFI     := REWRITE("RAMTEXT");
    .. Wir schreiben einen kurzen Text hinein...
    WRITELN(nFI, "Dies wird ein Text, ");
    WRITELN(nFI, "der anschließend an das Memo");
    WRITELN(nFI, "angehängt wird.");
    ...und schließen sie wieder
    CLOSE(nFI);
    .. RAMTEXT-Inhalt ans Memo anhängen
    ReadMemo(KUNDEN.Memo, "RAMTEXT", 1);
ENDPROC
```

Siehe auch: Clip2Text, Text2Clip

6.3.37 Statische Verknüpfung

Die statische Verküpfung entspricht einer Vorschrift, die zwei Tabellen miteinander verknüpft, indem nur diejenigen Datensatzkombinationen zugelassen werden, in der ein Feld (oder Ausdruck) der einen Tabelle mit dem Feld (oder Ausdruck) der anderen Tabelle übereinstimmt.

In einer Datenbank mit den Tabellen KUNDE und BESTELLG könnte eine Verknüpfung über die Kundennummer z.B. so aussehen:

```
$KUNDE[Nummer] = $BESTELLG[Kunden-Nummer]
```

Natürlichsprachlich übersetzt hieße diese Verknüpfung: "*Verknüpfe die Tabellen KUNDE und BESTELLG wobei das Feld Nummer (aus KUNDE) und das Feld Kunden-Nummer (aus BESTELLG) übereinstimmen müssen*"

Statische Verknüpfungen werden beim Verknüpfen von Tabellen, sowie in Datenbankjobs und Listen eingesetzt.

6.3.38 Steuerkommando

Steuerkommandos bestimmen die Art, wie ein Datenbankjob abgearbeitet wird. Sie bestehen immer aus zwei Zeichen, denen in den meisten Fällen eine Zahl als Argument nachgestellt wird. Mehrere Steuerkommandos können in einer Zeile zusammengefaßt werden, die einzelnen Kommandos werden durch Komma getrennt. Zum Beispiel: MT 5, PO 5, AK 0

6.3.39 Tabelle

Tabelle ist die Bezeichnung für die in einer Datei gespeicherten Datensätze. Eine Datenbank besteht meistens aus mehreren Tabellen, die durch Koppel- und Relationsfelder sowie statische Verknüpfungenverbunden sind. Das Datenfenster zeigt die Tabelle entweder in der Tabellensicht oder in der Formularsicht.

Durch die doppelte Verwendung des Begriffs in (Daten-)Tabelle und tabellarische Darstellung oder Tabellensicht kommt es manchmal zu Missverständnissen. Je nach Sinnzusammenhang kann sich der Ausdruck Tabelle also auf die eigentlichen Daten oder die Darstellung beziehen.

6.3.40 Tabellenfenster

Wenn Sie im Projektfenster eine Tabelle selektieren und 'Ausführen' anklicken, erhalten Sie eine editierbare Tabellensicht, wo Sie Daten ansehen und ändern können ohne ein Formular zu verwenden. Tabellenfenster stehen im User-Modus nicht zur Verfügung. Um im User-Modus eine Tabellensicht zu ermöglichen entwerfen Sie ein Formular und verwenden das Anzeigelelement *Tabelle*.

6.3.41 Transaktion

Unter einer *Transaktion* versteht man (im Datenbankbereich) die Überführung einer Tabelle von einem konsistenten Zustand in einen anderen. Während der Transaktion können inkonsistente Zustände auftreten. Wird beispielsweise ein neuer Datensatz aufgenommen, so wird zunächst dieser Datensatz eingefügt. jetzt ist die Tabelle solange in einem inkonsistenten Zustand, bis sämtliche Indexe und die Anzahl der Datensätze aktualisiert sind. Erst danach ist wieder ein konsistenter Zustand erreicht. Damit also keine inkonsistenten Zustände auftreten, muss eine Transaktion entweder komplett durchgeführt werden oder eben unterbleiben.

Siehe auch: Rollback

6.3.42 Unicode

Während die herkömmlichen Zeichenketten ein Byte pro Zeichen verwendeten und damit bis zu 255 verschiedenen Zeichen arbeiten konnten, werden bei Unicode-Zeichenketten zwei Byte pro Zeichen verwendet und somit sind bis zu 65767 verschiedene Zeichen möglich. Auf diese Weise gehören Schwierigkeiten mit Sonderzeichen und unterschiedlichen Sprachen der Vergangenheit an. Unicode ist der Standard unter Windows seit Windows NT 3.

6.3.43 User-Modus

Damit reine Anwender von *TurboDB Studio*-Projekten eine möglichst einfache Benutzeroberfläche vorfinden, kann das Programm im User-Modus gestartet werden, indem das Kommandozeilen-Argument /P oder /p beim Aufruf angegeben wird. Im User-Modus sind die Entwurfs-Fenster für Formulare, Berichte usw. nicht mehr zugänglich, auch das Projektfenster ist nicht sichtbar. Statt dessen hat der Anwender in einem einfachen, vollautomatisch erstellten Menü alle notwendigen Befehle zum Eingeben und Auswerten von Daten auf übersichtliche Weise zur Verfügung.

Der zuständige Programmteil für den User-Modus heißt Turbo-Applikationsprozessor und wird durch die erwähnten Kommandozeilen-Argumente aktiviert.

6.3.44 Vollständiger Name

Unter *TurboDB Studio* werden alle Projektelemente wie Formulare und Berichte einer Tabelle zugeordnet. Der vollständige Name eines solchen Projektelementes besteht im Namen der Tabelle gefolgt von einem Punkt und dem eigentlichen Namen des Elementes. Z.B. 'KFZ.Fahrzeugliste'.

6.3.45 Volltext-Index

Ein Volltextindex ist eigentlich gar kein Index im Sinne der relationalen Datenbanken. Während normale Indexe für jeden Datensatz der zugehörigen Tabelle einen Eintrag haben, besitzt ein Volltextindex variabel viele. Von überhaupt keinem bis einigen hunderten oder tausenden.

Ein Volltextindex ist eine Liste von Stichwörtern mit je einem Verweis auf den Datensatz, in dem das Stichwort vorkommt. Für diese Liste verwendet Turbo Datenbank eine eigene Tabelle. Zusätzlich wird noch eine Relations-Tabelle für die Verweise eingerichtet. Dieses Verfahren hat den Vorteil, dass Sie mit den herkömmlichen Verfahrensweisen von Turbo Datenbank auch die Stichwort-Tabelle und die Relations-Tabelle bearbeiten können.

6.3.46 Zugriff

Der Zugriff beschreibt, welche Datensätze der Tabelle in welcher Reihenfolge angezeigt werden. Der natürliche Zugriff ("Nummer") zeigt alle Datensätze in der Reihenfolge der Speicherung an, der Zugriff "Markierung" zeigt die markierten Datensätze in der Reihenfolge der Markierungsliste an und ein Index bestimmt die Reihenfolge durch sein Sortierkriterium.

6.4 Fehlermeldungen

6.4.1 Fehlermeldungen

Hier finden Sie eine Liste aller Fehlermeldungen, die beim Ausführen von Datenbankfunktionen oder beim Übersetzen und Ausführen von Makros, Modulen und Datenbankjobs vorkommen können.

6.4.2 1: Kann Datei "<Datei>" nicht öffnen

Bedeutung

Die Datei kann nicht geöffnet werden.

Ursachen

- Fehler bei Indexerstellung
- Indexbeschreibung zerstört
- Memodatei nicht zu öffnen
- Tabelle bereits geöffnet
- Falscher Code
- Zu viele Tabellen
- Speicher reicht nicht aus
- Ungültige Tabellenstruktur

Maßnahmen

Ermitteln Sie die genaue Ursache für diesen Fehler und beseitigen Sie diese.

6.4.3 2: Fehler beim Lesen aus Datei <Name>

Bedeutung

Beim Lesen von einem Datenträger ist ein Fehler aufgetreten.

Ursachen

Die Ursache dafür ist entweder, dass die Ressourcen für die Anwendung erschöpft sind oder dass die Netzwerkverbindung unterbrochen ist oder dass der Datenträger einen Fehler hat.

Maßnahmen

Erstellen Sie eine Sicherheitskopie der betroffenen Datenbank und der dazugehörenden Dateien.

Starten Sie den Rechner neu und versuchen Sie es nochmal.

Wenn der Datenträger-Zugriff über ein Netzwerk erfolgt, prüfen Sie, ob die Netzwerkverbindung noch in Ordnung ist.

Benutzen Sie System-Werkzeuge um die die betroffene Datei auf Fehler zu untersuchen.

6.4.4 3: Fehler beim Schreiben in Datei <Name>

Bedeutung

Beim Schreiben auf einen Datenträger ist ein Fehler aufgetreten.

Ursache

Dieser Fehler tritt auf, wenn das externe Speichermedium (Festplatte, Diskette, CD, DVD, Memory Stick) vollständig belegt oder defekt ist. Eine weitere Ursache für das Auftreten dieses Fehlers liegt

darin, dass von einem Makro oder Datenbankjob aus ein Satz geschrieben werden soll, die Tabelle aber im Nur-Lese-Modus geöffnet wurde.

Maßnahmen

Wechseln Sie in den Dateimanager und löschen Sie, was nicht mehr gebraucht wird. Anschließend sollten Sie die Datenbank überprüfen und etwaige Fehler durch Tabelle reparieren und Index wiederherstellen beseitigen.

6.4.5 4: Eintragen in Index <Name> nicht möglich

Bedeutung

Datensatz konnte nicht in den Index eingetragen werden.

Ursache

Der Datenträger ist voll oder defekt. Die Anwendung verfügt nicht über die benötigten Rechte.

Maßnahmen

Prüfen Sie, ob mit dem Datenträger alles in Ordnung ist und Sie über die nötigen Rechte auf die betroffene Datei verfügen.

Wenn der Index beschädigt ist, benutzen Sie die Funktion Index wiederherstellen im Index-Manager, um den Index zu reparieren.

6.4.6 5: Löschen aus Index <Name> nicht möglich

Bedeutung

Datensatz konnte nicht aus dem Index gelöscht werden.

Ursache

Der Datenträger ist voll oder defekt. Die Anwendung verfügt nicht über die benötigten Rechte. Es gibt auch Fälle, in denen der Index inkonsistent mit der Tabelle geworden ist.

Maßnahmen

Als erstes sollten Sie versuchen den Index mit der Funktion Index wiederherstellen aus dem Index-Manager zu reparieren. Prüfen Sie außerdem, ob mit dem Datenträger alles in Ordnung ist und Sie über die nötigen Rechte auf die betroffene Datei verfügen.

6.4.7 6: Index <Name> nicht vorhanden

Bedeutung

Der Index mit dem angegebenen Namen wird benötigt, existiert aber nicht.

Ursache

Tritt praktisch nur in TurboPL Makros und Datenbankjobs auf, wenn ein Indexname angegeben wird, der nicht (mehr) gültig ist.

Maßnahmen

Den korrekten Indexnamen verwenden.

6.4.8 7: Ungültige Suchbedingung

Bedeutung

Die angegebene Suchbedingung (Selektion) ist nicht korrekt.

Ursache

Dieser Fehler tritt beim Erstellen einer Tabelle oder beim Export auf, wenn eine dabei verwendete Suchbedingung (zum Beispiel Gültigkeitsbedingung oder Export-Selektion) nicht gültig ist. Dies liegt wiederum meistens an falsch geschriebenen Namen oder einer inkorrekten Syntax.

Maßnahmen

Überprüfen Sie die Suchbedingung und korrigieren Sie sie.

6.4.9 8: Import-Fehler

Bedeutung

Beim Importieren ist ein Fehler aufgetreten.

Ursachen

Die Struktur der zu importierenden Tabelle paßt nicht zu der Primärtabelle. Entweder ist ein Feld zuviel oder zu wenig oder die Typen sind nicht kompatibel. So können sie die Ursache näher einkreisen:

- Es wurde überhaupt kein Datensatz importiert. Hier können Sie davon ausgehen, dass die Primärtabelle mehr Felder hat, als die zu importierende.
- Es wurden einige Sätze importiert. Sie erhalten den zusätzlichen Hinweis "Falscher Typ". Hier passt das eingelesene Feld nicht zum entsprechenden der Primärtabelle. Häufige Ursachen hierfür sind:
- Illegale Datumsangaben wie 99-12-1 (muss entweder 1.12.99 oder 12/1/99 heißen)
- Leere Zahlenfelder (werden nur importiert, wenn U-Spezifikation angegeben wurde)
- Illegale Zahlenkonstanten, z.B. weil Buchstaben mit in der Zahl vorkommen
- Es wurde genau ein Satz importiert. Hier hat die einzulesende Tabelle wahrscheinlich mehr Felder als die Primärtabelle.

Maßnahmen

Passen Sie das Schema der Tabelle an die zu importierende Tabelle an.

6.4.10 9: Fehler im Ausgabeformat

Bedeutung

Das eingegebene Ausgabeformat entspricht syntaktisch oder semantisch nicht den Regeln.

Ursachen

Ausgabeformate werden ausschließlich in Datenbankjobs verwendet. Oft sind falsch geschriebene Bezeichner (Feldnamen, Variablen usw.) oder nicht korrekt eingesetzte Operatoren die Ursache für diese Fehlermeldung.

Maßnahmen

Korrigieren Sie das Ausgabeformat. Schlagen Sie den korrekten Aufbau gegebenenfalls in der Online-Hilfe nach.

6.4.11 10: Fehler beim Schließen der Tabelle <Name>

Bedeutung

Beim Schließen einer Tabelle ist ein Fehler aufgetreten

Ursache

Der Datenträger für die Tabelle ist entfernt worden oder defekt.

Maßnahmen

Stellen Sie den Datenträger wieder her und reparieren Sie anschließend die Datenbank.

6.4.12 11: Index-Information zu groß

Bedeutung

Es wurde eine Index-Definition angegeben, deren Schlüsselgröße die maximal zulässige Größe überschreitet.

Ursache

Bei Tabellen bis zu Level 3 darf die Index-Information aus maximal 255 Bytes bestehen.

Maßnahmen

Verkürzen Sie die Index-Definition des oder der String-Felder oder konvertieren Sie die Tabelle

nach Level 4.

6.4.13 12: Fehler beim Öffnen des Index <Name>

Bedeutung

Der Index konnte nicht geöffnet werden.

Ursache

Dieser Fehler wird angezeigt, wenn die betreffende Indexdatei entweder nicht gefunden werden kann oder eine Diskrepanz zwischen der Zahl der Indexeinträge und der Anzahl der Datensätze festgestellt wird (im Zusammenhang mit Fehler 13). Wird der Index nicht gefunden und handelt es sich um einen ADL-Index (INR, ID, IN1 bis IN9), so wird der Index nach der Bestätigung der Fehlermeldung automatisch neu aufgebaut. Erhalten Sie in der Folge den Fehler 28, so kann das ein Hinweis auf illegale Programm-Manipulationen sein. Auf jeden Fall ist dann die im Vorspann abgelegte Definition des ID-Index nicht mehr brauchbar. Einen letzten Rettungsversuch können Sie bei Fehler 28 nachlesen.

Maßnahmen

Prüfen Sie, ob die Datei für den Index noch existiert. Löschen Sie den Index in der Index-Verwaltung und erzeugen Sie ihn neu.

6.4.14 13: Index <Name> passt nicht zur Tabelle

Bedeutung

Die Anzahl der Einträge im Index stimmt nicht mit der Anzahl Zeilen in der Tabelle überein.

Ursache

Der Index wurde zum Beispiel durch einen Programmabsturz inkonsistent zur Tabelle.

Maßnahmen

Reparieren Sie den Index mit der Funktion Index wiederherstellen in der Index-Verwaltung.

6.4.15 14: <Name> ist kein Datenfeld

Bedeutung

Im SQL-Befehle, Makro oder Datenbankjob wird die Angabe eines Tabellenspalte erwartet aber nicht gefunden.

Ursache

Der Bezeichner ist falsch geschrieben oder das Datenfeld an dieser Stellen nicht zugreifbar.

Maßnahme

Prüfen Sie, ob das Datenfeld hier erlaubt und korrigieren Sie gegebenenfalls die Schreibweise.

6.4.16 16: Ungültige Tabellen-Verknüpfung

Bedeutung

Die Tabellen-Verknüpfung im Datenmodell oder in einem Relations-Kommando ist ungültig.

Ursache

Eine der an der Verknüpfung beteiligten Tabellen ist nicht (mehr) Teil der Datenbank oder kann nicht geöffnet werden. Die Verknüpfung enthält Schreib- oder Syntax-Fehler. Die in der Verknüpfung vorkommenden Tabellenspalten haben unterschiedlichen Typ.

Maßnahmen

Falls die Tabelle nicht Teil der Datenbank ist, können Sie die Verknüpfung entfernen. Andernfalls sorgen Sie dafür, dass die Tabelle auch geöffnet werden kann und die Verknüpfung korrekt formuliert ist.

6.4.17 19: Datensatz nicht gefunden

Bedeutung

Der gesuchte Datensatz wurde nicht gefunden.

Ursache

Es wurde nach einem Datensatz gesucht, der so nicht existiert. Zum Beispiel überschreitet die Satznummer die Anzahl an Datensätzen in der Tabelle.

Maßnahmen

Das ist kein wirklicher Fehler und muss deshalb nicht beseitigt werden.

6.4.18 20: Modul <Name> nicht gefunden

Bedeutung

Das Modul ist im übersetzten Programm des Projektes nicht enthalten.

Ursache

In einer uses-Anweisung oder in einem ExecMacro-Aufruf wurde ein Modul angegeben, das nicht im Projekt enthalten ist oder noch nicht übersetzt wurde.

Maßnahme

Binden Sie das Modul mit in das Projekt ein und übersetzen Sie alles neu. Da ExecMacro seit TurboDB Studio überflüssig ist, sollten Sie es durch einen direkten Aufruf der Prozedur ersetzen.

6.4.19 22: Indexdatei <Name> existiert bereits

Bedeutung

Eine Index-Datei mit diesem Namen ist schon vorhanden.

Ursache

Sie versuchen, einem benutzerdefinierten Index einen Namen zu geben, den schon eine andere Indexdatei im gleichen Verzeichnis trägt.

Maßnahmen

Anderen Namen für den Index wählen.

6.4.20 23: Index <Name> bereits geöffnet

Bedeutung

Index kann wegen eines Namens-Konflikts nicht angelegt werden.

Ursache

Die Tabelle enthält schon einen Index mit diesem Namen.

Maßnahmen

Einen anderen Namen für den Index vergeben.

6.4.21 25: Syntax-Fehler

Bedeutung

Der TurboPL-Ausdruck ist nicht korrekt gebildet.

Ursache

Hier gibt es so viele mögliche Ursachen, wie Möglichkeiten einen Programmausdruck falsch zu schreiben. Fehlende oder falsch getippte Schlüsselwörter, fehlende oder überzählige Semikolons, fehlende oder überflüssige Operatoren usw.

Maßnahmen

Der fehlerhafte Ausdruck wird angezeigt, überprüfen Sie ihn genau auf Tipp- und sonstige Fehler.

6.4.22 26: Ungültiger Dateiname

Bedeutung

Der angegebene Dateiname entspricht nicht den Regeln.

Ursache

Der Dateiname ist leer oder enthält ungültige Zeichen.

Maßnahme

Wählen Sie einen anderen Dateinamen.

6.4.23 28: Fehlerhafte Definition für Index <Name>

Bedeutung

Eine gespeicherte Index-Definition ist inkonsistent zur Tabelle.

Ursache

Durch einen Absturz oder eine defekte Datei kann die Index-Definition nicht mehr benutzt werden.

Maßnahmen

Versuchen Sie, die Tabelle mit der Funktion Tabelle reparieren oder der Funktion Index wiederherstellen wieder in Gang zu bringen.

6.4.24 29: Falsche Programmversion

Bedeutung

Die Version des Programms in der prg-Datei stimmt nicht mit der Version des Projektes überein.

Ursache

Das Programm wurde seit dem letzten Update von TurboDB nicht neu übersetzt. Das Programm wurde mit einer anderen Version übersetzt und dann kopiert.

Maßnahmen

Übersetzen Sie das Programm mit aktuellen Version von TurboDB neu.

6.4.25 30: Datensätze müssen bezüglich <Feldliste> eindeutig sein

Bedeutung

Es wurde versucht einen Datensatz zu ändern oder neu einzutragen, der eine Eindeutigkeits-Regel verletzt.

Ursache

Sie wollen einen Datensatz neu eingeben, dessen Inhalt dazu führt, dass ein eindeutiger Index zwei gleiche Einträge enthalten würde. Angenommen, Sie haben einen Index über Name erstellt und diesen als eindeutig festgelegt (was eindeutig keine gute Idee ist). Solange nur ein Müller in der Datenbank ist, geht alles gut, beim zweiten erhalten Sie diese Fehlermeldung.

Maßnahme

Ändern Sie den Datensatz oder überdenken Sie noch einmal die Eindeutigkeits-Regel.

6.4.26 31: Memo-Datei <Name> kann nicht geöffnet werden

Bedeutung

Die Datei mit den Memo-Einträgen lässt sich nicht öffnen.

Ursache

Die Datei ist entweder nicht vorhanden oder defekt. Die Anwendung besitzt nicht die nötigen Rechte an dieser Datei.

Maßnahme

Prüfen Sie, ob die Datei beim Kopieren vergessen worden ist und ob alle nötigen Rechte

vorhanden sind. Wenn das der Fall ist, versuchen Sie mit der Funktion Tabelle reparieren, die Tabelle wieder in Ordnung zu bringen.

Für Spezialisten hier ein Rettungsversuch für den Fall, dass die MMO-Datei vorhanden und die Tabelle nicht verschlüsselt ist: Ändern Sie mit einem Hex-Editor die ersten 8 Bytes der MMO-Datei auf 0. Öffnen Sie die Tabelle und exportieren Sie sie in eine neue Tabelle. Löschen Sie die Ursprungstabelle und arbeiten Sie mit der Kopie weiter.

Andernfalls müssen Sie auf den Memoinhalt verzichten. Hier ein Trick, wie Sie wenigstens die zugehörige Tabelle retten können:

1. Erzeugen Sie eine neue Tabelle mit der gleichen Struktur. Geben Sie dabei den gleichen Verschlüsselungscode wie bei der Ursprungstabelle an.

2. Wechseln Sie in den Explorer. Falls die defekte Memodatei noch vorhanden ist, löschen Sie diese.

3. Erzeugen Sie eine Kopie der neu generierten Memodatei mit dem Namen der defekten Memodatei.

4. Wechseln Sie in TurboDB Studio zurück und öffnen Sie die Ursprungstabelle. Exportieren Sie die Tabelle in einem Text-Format. Schließen Sie die Tabelle.

5. Öffnen Sie die neu generierte Tabelle und importieren Sie den zuvor exportierten Datenbestand.

6.4.27 32: Memo- und Blobfelder nicht zulässig

Bedeutung

In einem Ausdruck wurde ein Memofeld oder ein Blobfeld angegeben, wo dies nicht erlaubt ist.

Ursache

Memo- und Blobfelder können an einigen Stellen nicht verwendet werden, wo Datenfelder erwartet werden, zum Beispiel

- Als Elemente in einer Index-Definition
- Direkt in Ausdrücken und Suchbedingungen
- Als Argumente von Aggregationsfunktionen

Maßnahmen

Sie müssen den Ausdruck ohne Verwendung von Memo- und Blobfeld formulieren. Oftmals hilft hier die Funktion MemoStr weiter.

6.4.28 33: Fehler beim Schreiben des Memos/Blobs

Bedeutung

Die Memo- oder Blob-Daten konnten nicht auf den Datenträger geschrieben werden.

Ursache

Meist ist hier entweder der Datenträger voll oder das Memo hat die maximale Größe von 4 GB erreicht. Es könnte auch sein, dass der Datenträger defekt oder die Netzwerkverbindung unterbrochen ist.

Maßnahmen

Sichern Sie den aktuellen Datenstand und untersuchen Sie, welche der obigen Ursachen zutreffend ist.

6.4.29 34: Berechtigung für diese Operation nicht vorhanden

Bedeutung

Es wurde versucht, eine Funktion auszuführen, für die gegenwärtig keine Berechtigung existiert.

Ursache

Im Wesentlichen gibt es drei Ursache für diese Fehlermeldung:

- Es wurde versucht, einen Datensatz anzuhängen, zu bearbeiten oder zu löschen, obwohl das entsprechende Recht nicht vorhanden ist.

- Es wurde versucht, einen Index neu zu erstellen, obwohl das entsprechende Recht nicht vorhanden ist.
- Es wurde versucht, einen System-Index (*.inr, *.ind, *.in1 usw.) zu bearbeiten oder zu löschen

Maßnahme

Wenn Sie die Berechtigung nicht besitzen, können Sie die Operation nicht durchführen. Die andere Möglichkeit ist natürlich, die Berechtigung einzuräumen.

6.4.30 35: Tabelle <Name> bereits geöffnet

Bedeutung

Es wurde versucht, eine Tabelle zu öffnen, die entweder selbst schon geöffnet ist oder die den selben Namen hat wie der einer schon geöffneten Tabelle.

Ursache

Dieser Fehler tritt auch bei Operationen auf, die auf den ersten Blick nichts mit dem Öffnen einer Tabelle zu tun haben, wie zum Beispiel beim Löschen oder Umbenennen einer Tabelle oder beim Exportieren in eine Tabelle. Das liegt daran, dass bei diesen Operationen die Tabelle intern zuerst geöffnet werden muss. Bei der Tabelle kann es sich auch um eine indirekt geöffnete Tabelle handeln, wie zum Beispiel um eine Relationstabelle.

Maßnahmen

Wenn der Fehler in einem Programm mit OpenDb auftritt, können Sie zuvor mit FindTable prüfen, ob die Tabelle schon offen ist. Andernfalls müssen Sie entscheiden, ob eine der beiden Tabellen umbenannt werden kann oder ob die erste Tabelle geschlossen wird, bevor die zweite geöffnet wird.

6.4.31 36: <Zugriff> ist kein gültiger Zugriff für die Tabelle

Bedeutung

Der angegebene Zugriff ist ungültig.

Ursache

Das Argument von *setAccess* im Datenbankjob ist weder "Nummer" noch "Markierung" noch ein Index der Tabelle. Beachten Sie, dass Sie bei Id- und Inr-Indexen die Endung mit angeben müssen.

6.4.32 37: Datei <Name> nicht gefunden

Bedeutung

Eine Datei auf dem Datenträger soll geöffnet werden, ist aber nicht vorhanden.

Ursache

Der Dateiname ist falsch angegeben oder die Datei wurde versehentlich gelöscht. Es kann auch sein, dass die Datei nicht in dem Verzeichnis gesucht wird, das Sie annehmen. Das ist bei den Anweisungen uses und include manchmal der Fall.

Maßnahmen

Korrigieren Sie den Dateinamen oder stellen Sie die fehlende Datei wieder her.

6.4.33 38: Die angegebenen Zugangsdaten sind ungültig

Bedeutung

Das Passwort und/oder der Code für die Tabelle sind nicht richtig angegeben worden.

Ursache

Wenn Sie eine Tabelle öffnen wollen, für die Passwort und Verschlüsselungscode vereinbart wurden, und Sie diese Informationen nicht haben, so handelt es sich weniger um eine Fehlermeldung als vielmehr um den Hinweis, Ihre Finger davon zu lassen.

In Tabellen bis einschließlich Level 3 kann diese Fehlermeldung auch bedeuten, dass Daten in der Tabelle zerstört sind.

Maßnahmen

Geben Sie die korrekten Zugangsdaten für die Tabelle an. Wenn dieser Fehler trotzdem gemeldet wird und die Tabelle den Level 3 oder kleiner hat, müssen Sie versuchen, sie mit der Funktion Tabelle reparieren in Ordnung zu bringen.

6.4.34 41: Ausdruck nicht vollständig

Bedeutung

Ein Ausdruck oder eine Suchbedingung ist nicht vollständig.

Ursache

Meistens handelt es sich hier um reine Tippfehler.

Maßnahme

Korrigieren Sie den Ausdruck.

6.4.35 42: Operator passt nicht zu Operand

Bedeutung

Der verwendete Operator ist für die angegebenen Datentypen nicht zugelassen.

Ursache

Sie können nicht alle Operatoren mit allen Datentypen kombinieren, zum Beispiel können Sie Zeichenketten nicht subtrahieren und Zahlen nicht mit ähnlich vergleichen.

Maßnahmen

Korrigieren Sie den Ausdruck. Eventuell müssen Sie in der Referenz nach dem richtigen Operator oder einer passenden Funktion für die gewünschte Berechnung suchen.

6.4.36 43: Der Wert ist außerhalb des erlaubten Bereichs

Bedeutung

Innerhalb einer Berechnung werden die Zahlen größer als technisch möglich ist.

Ursache

Die Zahlenbereich der TurboPL-Datentypen sind sehr groß, so dass dieser Fehlermeldung meistens ein Schreibfehler zugrunde liegt. Es kann auch sein, dass durch eine ungünstige Berechnungsreihenfolge der Zahlenbereich überschritten wird.

Maßnahmen

Durch Korrektur oder Umstellung des Ausdrucks lässt sich dieser Fehler in den allermeisten Fällen beheben. Ansonsten bleibt nur, den Fehler abzufangen (in einem TurboPL-Programm oder Datenbankjob zum Beispiel mit try/except) und geeignet zu reagieren.

6.4.37 44: Typen stimmen nicht überein: <Typ1> -> <Typ2>

Bedeutung

Es werden zwei Werte verglichen, verrechnet oder zugewiesen die nicht miteinander kompatibel sind.

Ursache

Dies ist ein häufiger Programmierfehler, der durch verwechselte Feldnamen, falsche Variablentypen und ähnliches entsteht.

Maßnahmen

Meistens genügt es den Ausdruck richtigzustellen. Wenn die Typen tatsächlich nicht kompatibel sind, hilft der Einsatz einer Konvertierungsfunktion (zum Beispiel *Str* oder *DateStr* oder *DateTimeVal*) oder einer Typ-Konvertierung mit *as* (*TurboPL*) beziehungsweise *CAST* (SQL).

6.4.38 45: Die Zeichenfolge <Zeichen> ist hier nicht erlaubt

Bedeutung

In einem Ausdruck ist eine an dieser Stelle nicht zulässige Zeichenfolge enthalten.

Ursache

Meist ist ein Schreibfehler die Ursache für diese Fehlermeldung.

Maßnahmen

Korrigieren Sie den Ausdruck.

6.4.39 46: "%s" ist keine Zahl

Bedeutung

An dieser Stelle wird eine Zahl erwartet, aber die angegebene Konstante ist weder eine Ganz- noch eine Fließkommazahl

Ursache

Meist ist ein Tippfehler der Grund für diese Fehlermeldung. Beachten Sie aber, dass in TurboPL-Ausdrücken der Punkt als Dezimaltrenner verwendet wird, dass ein Datum in der Form T.M.JJJJ (zum Beispiel 30.3.2004) angegeben wird und eine Uhrzeit als HH:NN:SS (zum Beispiel 23:49:12).

Maßnahmen

Korrigieren Sie den Ausdruck.

6.4.40 48: Logischer Operand fehlt

Bedeutung

TurboDB Studio erwartet nach einem logischen Operator (Vergleichsoperator wie kleiner, bis etc.) einen Ausdruck, kann jedoch keinen finden.

Ursache

Wahrscheinlich ein Tippfehler.

Maßnahmen

Korrigieren Sie den Ausdruck.

6.4.41 49: Das Argument ist außerhalb des erlaubten Bereichs

Bedeutung

Eine Prozedur wird mit einem ungültigen Wert aufgerufen, beispielsweise die Prozedur *ReadRec* mit der Satznummer -13 oder die Prozedur *High* mit einer nicht vorhandenen Array-Dimension.

Ursache

Das Programm überprüft Benutzereingaben nicht oder unvollständig. Ein anderer Programmteil liefert ein falsches Ergebnis.

Maßnahmen

Entweder Sie ändern das Programm so ab, dass eine Übergabe von ungültigen Argumenten nicht mehr vorkommen kann, oder Sie reagieren auf diese Fehlermeldung in Ihrem Programm durch Einfügen einer Systemfehlerbehandlung mit *try* und *except*.

6.4.42 50: Unbekannter Bezeichner <Name>

Bedeutung

Bei dem eingegebenen Wort handelt es sich weder um Datenfeld, eine Variable, einen Prozedur- oder Funktionsnamen, noch um eines der reservierten Wörter von TurboPL. Beachten Sie bitte, dass bei sämtlichen selbstdefinierten Bezeichnern (Feldnamen, Variable, Prozeduren etc.) die Groß/Kleinschreibung beachtet werden muss. Die Meldung enthält auch den Bezeichner, der nicht

gefunden wurde.

Ursache

Meistens ist der Bezeichner einfach nur falsch geschrieben.

Maßnahmen

Korrigieren Sie den Bezeichner.

6.4.43 51: Array-Variable erwartet

Bedeutung

Die Prozedur erwartet einen Parameter, der ein Array ist; ein solcher wurde aber nicht angegeben. Beispiele für solche Prozeduren sind *StrSort* und *High, aber auch Benutzer-definierte Prozeduren können Array-Parameter definieren.*

Ursache

Wahrscheinlich haben Sie entweder die aufgerufene Prozedur oder die Variable verwechselt.

Maßnahme

Korrigieren Sie den Prozedur-Aufruf.

6.4.44 52: Unbekannter Fehler

Bedeutung

Hier kann der Fehler nicht näher spezifiziert werden.

Ursache

Es ist ein Fehler in einem Bereich aufgetreten, in dem eigentlich kein Fehler erwartet wird.

Maßnahmen

Versuchen Sie durch Rücknahme der letzten Änderungen den Fehler zu eliminieren. Melden Sie sich beim dataWeb-Support und berichten Sie den genauen Hergang dieses Fehlers.

6.4.45 53: Zu viele Variablen

Bedeutung

In einem TurboPL-Programm sind insgesamt bis zu 2048 Variablen möglich. Diese Höchstgrenze wurde überschritten.

Ursache

Ihr Programm verwendet zu viele globale Variablen. Zur Laufzeit eines Programms werden die lokalen Parameter einer Prozedur bzw. Funktion ebenfalls als Variable angelegt, wenn diese aufgerufen wird. Bei der Beendigung einer Prozedur werden diese Variablen wieder freigegeben. Der Fehler kann somit auch auftreten, wenn eine Prozedur mit vielen lokalen Variablen oftmals rekursiv aufgerufen wird.

Maßnahmen

Wenn Sie tatsächlich über 1000 globale Variablen verwenden, sollten Sie überlegen, ob Sie diese nicht zu lokalen Variablen machen können oder in eine Datenbanktabelle auslagern.

Wenn das Problem daher kommt, dass Sie zum Beispiel eine Prozedur mit 10 lokalen Variablen 205 mal rekursiv aufrufen, müssen Sie versuchen die Anzahl der Variablen zu reduzieren, indem Sie weniger zwischenspeichern, lokale Variable zu globalen machen oder die Daten in eine Datenbanktabelle auslagern.

6.4.46 54: "=" fehlt

Bedeutung

Es wird im Ausdruck ein Gleichheitszeichen erwartet aber ein anderes Zeichen gefunden.

Ursache

Eine Variablenzuweisung muss immer die Form *Let Name = Ausdruck* haben. Vor allem dürfen

zwischen dem Name und dem Gleichheitszeichen höchsten Leerzeichen stehen. Folgende Zuweisung führt deshalb zu einem Fehler: *Let Name := Ausdruck*. Der Fehler kann auch in einem Relationskommando auftreten, wenn Sie eine virtuelle Tabelle definieren wollen (Beispiel *FIRMEN1 := FIRMEN* statt *FIRMEN1 = FIRMEN*).

Maßnahmen

Korrigieren Sie den Ausdruck

6.4.47 55: Zahl-Konstante erwartet

Bedeutung

Im Ausdruck muss an dieser Stelle eine Zahl stehen, ein Ausdruck ist nicht erlaubt.

Ursache

Dieser Fehler tritt meist im Zusammenhang mit einer Formatieranweisung auf. Nach dem Doppelpunkt muss eine Zahl stehen, Variable oder sonstige Ausdrücke sind nicht zugelassen. Also nicht NAME:Breite sondern NAME:30. Falls Sie wirklich eine dynamische Formatierung benötigen, können Sie auf die verfügbaren Stringfunktionen zurückgreifen.

Maßnahmen

Ersetzen Sie den Ausdruck durch eine Zahl.

6.4.48 56: <Zeichenfolge> erwartet

Bedeutung

Im Ausdruck muss an dieser Stelle die angegebene Zeichenfolge stehen, tatsächlich wurde aber etwas anderes gefunden.

Ursache

Meist ein Schreibfehler.

Maßnahmen

Korrigieren Sie den Ausdruck.

6.4.49 57: Spaltenanzahl passt nicht

Bedeutung

Es wurden entweder zu wenig oder zu viele Spalten angegeben. Dies tritt zum Beispiel im SQL INSERT-Befehl auf, wenn die Anzahl der Elemente in VALUES nicht übereinstimmt mit der Anzahl der Elemente in der Spaltenliste.

Ursache

Wahrscheinlich haben Sie sich einfach nur verzählt.

Maßnahmen

Bringen Sie die Liste der Spalten in Übereinstimmung.

6.4.50 58: Tabelle <Name> nicht gefunden

Bedeutung

Es wurde ein Tabellenname angegeben, der nicht existiert.

Ursache

Sie haben sich vertippt. Die Tabelle war einmal Bestandteil der Datenbank, ist es jetzt aber nicht mehr oder wurde umbenannt.

Maßnahmen

Passen Sie den Tabellennamen an.

6.4.51 59: Zu viele Tabellenspalten

Bedeutung

Es wurde versucht eine Tabelle mit mehr als 1000 Spalten zu definieren.

Ursache

Wahrscheinlich liegt ein ungünstiger Datenbank-Entwurf vor.

Maßnahmen

Speichern Sie unterschiedliche Entitäten in verschiedenen Datenbanktabellen, zum Beispiel sollten die Daten eines Wagens und die seines Käufers nicht in der selben Tabellen stehen sondern in zwei verschiedenen. Wenn Sie dann immer noch zu viele Spalten haben, können Sie diese trotzdem in zwei Tabellen unterbringen und diese mit einer 1:1-Beziehung verknüpfen.

6.4.52 60: Ausdruck zu komplex

Bedeutung

Der Ausdruck enthält eine Anzahl von Verschachtelungs-Ebenen, die nicht mehr zugelassen ist.

Ursache

Der Ausdruck verstößt gegen eine der folgenden Obergrenzen:

- In einer Koppelfeldnotation darf nur die erste Beziehung eine Relation sein.
- Eine Koppelfeldnotation darf nicht mehr als 15 Stufen umfassen.
- Ein hierarchischer Index kann maximal 10 Stufen enthalten.
- Die übersetzte Form eines berechneten Index kann bis zu 39 Zeichen enthalten.
- Die Anzahl der möglichen rekursiven Prozeduraufrufe ist durch den Speicherbedarf begrenzt (jedoch sehr, sehr hoch)

Maßnahmen

Wenn die Fehlermeldung durch eine rekursive Prozedur hervorgerufen wird, war die hohe Rekursionstiefe wohl nicht beabsichtigt und kann korriert werden. In allen anderen Fällen müssen Sie den Entwurf der Datenbank überdenken.

6.4.53 63: Tabelle <Name> kann nicht geöffnet werden

Bedeutung

Die Tabelle ist zwar vorhanden, kann aber nicht geöffnet werden.

Ursache

Die Rechte zum Öffnen der Tabellendatei fehlen. Die Tabelle ist von einer anderen Anwendung exklusiv geöffnet worden.

Maßnahmen

Verschaffen Sie sich die Rechte auf die Dateien und sorgen Sie dafür, dass auf gemeinsam benutzte Datenbestände nur im gemeinsamen Modus zugegriffen wird.

6.4.54 64: Tabelle enthält keine Auto-Nummer-Spalte

Bedeutung

Bestimmte Operationen sind mit einer Tabelle nur möglich, wenn sie ein Feld mit Auto-Nummer enthält, zum Beispiel können nur solche Tabellen über Koppel- und Relationsfelder verknüpft werden.

Ursache

Im Schema für die Tabelle ist keine Spalte mit Auto-Nummer vorhanden.

Maßnahme

Fügen Sie eine Auto-Nummer-Spalte zur Tabelle hinzu.

6.4.55 67: Zu viele Indexe

Bedeutung

Diese Meldung erhalten Sie, wenn bei Level 3 die maximale Anzahl von 16 Indexen pro Tabelle überschritten wird.

Ursache

Dies kann z.B. passieren, wenn

- eine Tabelle mehr als 9 Koppelfelder ohne U-Spezifikation enthält, oder
- in einem Modul oder Datenbankjob die Funktion GenIndex aufgerufen wird, obwohl schon 15 Indexe zu einer Tabelle angelegt wurden.

Maßnahmen

Entfernen Sie weniger wichtige Indexe aus dem Schema. Richten Sie wo möglich Koppelfelder ohne Rückkopplung ein. Konvertieren Sie die Tabelle in Level 4, um beliebig viele Indexe anlegen zu können.

6.4.56 71: Bezeichner <Name> doppelt definiert

Bedeutung

In einem Modul oder Datenbankjob darf jede Variable nur einmal mit *vardef* in einem Gültigkeitsbereich definiert werden. Die Parameterangaben einer Prozedur werden wie *vardef* behandelt. In folgendem Beispiel sind die Bezeichner A und C doppelt definiert (nicht aber B und D):

```
vardef B, D: Real;
procedure Test(A, B, C: Real)
    vardef A, C, D: Real
endproc
```

Ursache

Ein ungünstig gewählter Variablen oder Parametername.

Maßnahmen

Verwenden Sie einen anderen Namen für die Variable.

6.4.57 72: Relations-Tabelle <Name> existiert bereits

Bedeutung

TurboDB Studio bildet den Namen von Relations-Tabellen aus dem Namen des Relationsfeldes. Damit es zu keinem Datenverlust kommt, prüft das Programm, ob eine solche Tabelle schon existiert. Im positiven Fall erhalten Sie die obige Meldung.

Ursache

Eine andere Tabelle mit einem Relationsfeld des gleichen Namens befindet sich bereits im Verzeichnis.

Maßnahmen

Falls Sie die vorhandene Relations-Tabelle nicht mehr benötigen, können Sie inklusive der beiden Index-Dateien IN1 und IN2 mit dem Explorer löschen. Ansonsten müssen Sie das Relationsfeld umbenennen, weil sich damit auch der Name der Tabellen-Datei ändert.

6.4.58 73: Zu viele Koppelfelder

Bedeutung

Eine Tabelle kann nur höchstens 9 Koppelfelder enthalten. Diese Grenze wurde überschritten.

Ursache

Der Datenbank-Entwurf ist zu kompliziert.

Maßnahmen

Sie können die Tabelle in mehrere aufspalten, um die Anzahl an Koppelfeldern zu erhöhen.

6.4.59 74: Nur 15 Stellen erlaubt

Ursache

Eine Fließkommazahl im Programm kann nur mit bis zu 15 Gesamtstellen angegeben werden.

6.4.60 75: Zeichenkette zu lang

Ursache

Es wurde eine Zeichenkette mit mehr als 2 Mrd. Zeichen definiert oder übergeben.

6.4.61 81: Bereichsüberlauf

Ursache

Dieser Fehler tritt in Datenbankjobs oder Modulen auf, wenn der Index eines Arrays über die definierten Grenzen hinausläuft.

Beispiel:

```
vardef Namen: String[10];
Namen[11] := "Hans";
```

6.4.62 82: Illegales Kommando

Ursache

In einem Modul oder Datenbankjob trifft der Compiler auf ein Kommando, mit dem er (zumindest im aktuellen Zusammenhang) nichts anfangen kann. Der Fehler wird beispielsweise auch bei *until* gemeldet, wenn zuvor kein *repeat*-Kommando festgelegt wurde.

6.4.63 86: Illegale Textaufteilung

Ursache

In Datenbankjobs, Reports und Serienbriefen müssen Sie immer folgende Reihenfolge der Bereiche einhalten: Prolog, Kopf, Fuß, Gruppe,Gruppe, Daten, Epilog. Es dürfen zwar beliebig viele Bereiche wegfallen, jedoch darf die Reihenfolge der verbleibenden Bereiche nicht verändert werden. Zudem darf jeder Bereich nur einmal festgelegt werden.

Wenn Sie den Modul-Typ angeben, muss dies zwingend in der der ersten Zeile geschehen.

6.4.64 87: Ungültiger Zugriff

Ursache

Für diesen Fehler gibt es zwei mögliche Ursachen:

1. In einem Modul oder Datenbankjob greifen Sie über eine der Indexfunktionen auf einen nicht vorhandenen Index zu oder benützen die Funktion *FindRec* in der Kurzform, ohne vorher den Zugriff auf einen Index gesetzt zu haben.
2. In eine Datenbankjob verwenden Sie das Kommando *Replace*, und eine der Ersetzungen tangiert den aktuellen Index. Abhilfe in diesem Fall: Zugriff bei Jobs mit *Replace* immer auf Nummer setzen.

6.4.65 88: Tabelle nicht geöffnet

Ursache

Es wird versucht auf eine nicht geöffnete Tabelle zuzugreifen. Dieser Fehler tritt vor allem in Modulen auf, die Sie ohne Neuübersetzung starten und vorher eine oder mehrere Tabellen aus dem Projekt entfernt haben. Abhilfe: Neuübersetzung des Moduls.

6.4.66 89: Variable erwartet

Ursache

Einige Standardfunktionen wie *NLoop* erwarten als Parameter den Namen einer Variablen (also eine Variablen-Referenz). An dieser Stelle dürfen Sie keinen Ausdruck übergeben. Auch bei selbstdefinierten Prozeduren, für die Sie einen oder mehrere Referenzparameter festlegen, dürfen später an diesen Stellen nur Variablennamen übergeben werden.

6.4.67 90: Fehler beim Schreiben des Index

Ursache

Ein Datensatz konnte nicht korrekt in die Indexdatei eingefügt werden, weil entweder der Index logisch defekt oder die Platte voll ist. Im ersten Fall hilft Wiederherstellen des Index, im zweiten das Aufräumen der Platte.

6.4.68 91: Fehler beim Lesen des Index

Ursache

Eine Indexinformation konnte in einer Indexdatei nicht gefunden werden. Auch hier können logische Fehler oder Hardwareprobleme die Ursache sein. In den meisten Fällen ist nach dem Wiederherstellen aller Indexe wieder alles in Ordnung.

Die Hauptursache für diesen Fehler liegt darin, dass das Projekt beim letzten mal nicht ordnungsgemäß geschlossen wurde.

6.4.69 92: Ende des Unterreports nicht gefunden

Ursache

Ein mit *sub* begonnener Unterreport hat kein zugehöriges *endsub*. Der Cursor steht am zugehörigen *Sub*-Kommando.

6.4.70 93: Kommando in diesem Bereich nicht erlaubt

Ursache

Vor allem im Datenbankjob ist nicht jedes mögliche Kommando an jeder Stelle zulässig. Zum Beispiel muss das Kommando *setSortOrder* im Prolog stehen und darf nicht im Datenbereich aufgerufen werden.

Maßnahme

Lesen Sie die Dokumentation zu dem Kommando und positionieren Sie es an die richtige Stelle.

6.4.71 94: Es können nicht mehr als 255 Tabellen geöffnet werden

Ursache

Die Anwendung versucht, auf mehr als 255 Tabellen gleichzeitig zuzugreifen. Das ist nicht möglich. Schließen Sie nicht mehr benötigte Tabellen, bevor sie weitere Tabellen öffnen.

6.4.72 95: Index defekt

Ursache

Der Fehler zeigt zirkuläre Referenzen innerhalb des Index an. Das kann vorkommen, wenn Neueingabe, Editieren oder Löschen aus irgendeinem Grund nicht vollständig durchgeführt werden konnten (Speichermangel, Stromausfall).

Maßnahme

Reparieren Sie den Index, zum Beispiel mit dem *TurboDB Viewer* unter *Table/Maintain...*, dem

TurboDB Studio Datenmodellfenster, oder der Methode *UpdateIndex* in der VCL-Komponentenbibliothek.

6.4.73 97: Der Block "<Kommando>" wurde nicht korrekt abgeschlossen.

Ursache

Alle Block-Kommandos müssen mit dem entsprechenden Befehl abgeschlossen werden. solange mit ende, *for* mit *next*, *if* mit *end*, *try* mit *except* usw.

6.4.74 98: Fehlerhafte Indexdefinition

Ursache

Der Fehler wird gemeldet, wenn sie eine Indexdefinition angeben und diese nicht den syntaktischen und semantischen Regeln entspricht. Hinzu kommt die Möglichkeit, dass bei einem sortierten Sub-Report zwar die Indexdefinition korrekt ist, aber im Sub-Report relationale Zugriffe erfolgen. Ein Sortierkriterium (=Indexdefinition) kann nur angegeben werden, wenn sich der Sub-Report auf eine Tabelle (zusätzlich zur äußeren Primärtabelle) bezieht.

Falls Sie jedoch ganz sicher sind, dass bei einem Sub-Report über mehrere Tabellen jeweils maximal eine Datensatzkombination existiert (also nur n:1-Verbindungen), so können Sie die strenge Regel folgendermaßen unterlaufen: Sie definieren das Sortierkriterium als parameterlose Funktion mit *DEF*, geben diese Funktion (in Klammern) als Sortierkriterium an und schalten schließlich noch die Fehlerbehandlung mit *.EC 1* ab.

6.4.75 99: Speicher reicht nicht aus

Ursache

Für die gewünschte Aktion ist nicht genug Speicher vorhanden. Dies kann auch daran liegen, dass im *TurboPL*-Programm eine endlose Rekursion aufgetreten ist.

6.4.76 100: Demoversion erlaubt nicht mehr Datensätze

Ursache

Sie arbeiten mit einer Demoversion von *TurboDB*, die nicht mehr Datensätze pro Tabelle zulässt, als schon eingetragen sind.

6.4.77 101: Prozeduren dürfen nicht verschachtelt werden

Ursache

Selbstdefinierte Prozeduren und Funktionen dürfen nicht verschachtelt werden. Das bedeutet, dass zwischen *procedure* und *endproc* kein weiteres *procedure*-Kommando stehen darf.

6.4.78 102: endproc fehlt

Ursache

Sie haben eine Prozedur bis zum Programmende nicht mit *endproc* abgeschlossen.

6.4.79 103: Tabelle wird noch von anderen Anwendungen benutzt

Ursache

Wenn noch andere Sitzungen auf die Tabelle zugreifen, sind bestimmte Aktionen, zum Beispiel Restrukturieren, Löschen und Umbenennen nicht zugelassen.

6.4.80 104: Netzoperation abgebrochen

Bedeutung

Eine Operation auf einer Tabelle kann nicht durchgeführt werden, weil die Tabelle gesperrt ist.

Ursache

Eine andere Anwendung greift auf die selbe Tabelle zu und benutzt sie gerade oder hält sie gesperrt.

6.4.81 105: Datensatz wird bereits editiert

Bedeutung

Der Datensatz kann nicht editiert oder gelöscht werden, weil er von einer anderen Anwendung gerade schon editiert wird.

Ursachen

Eine weiter Anwendung greift auf die selbe Tabelle zu und editiert diesen Datensatz.

6.4.82 106: Fehler beim Einloggen

Bedeutung

Beim Öffnen einer Tabelle im Mehrplatz-Modus konnte die notwendige net-Datei nicht geöffnet und nicht erzeugt werden.

Mögliche Ursachen

1. Der Anwender darf im Verzeichnis der Tabelle keine Dateien erzeugen, öffnen, schreiben oder sperren.
2. Der Zugriff auf die net-Datei ist nicht möglich, weil gerade eine andere Anwendung diese ungewöhnlich lange physikalisch sperrt.

6.4.83 107: Ungültige Verbindungs-Id

Bedeutung

Die angegebene Rechner-Nummer ist ungültig

6.4.84 109: Unbekannter Fehler im Netz

Bedeutung

TurboDB hat entdeckt, dass die net-Datei ungültig ist.

Maßnahmen

Schließen Sie sofort alle Anwendungen, die auf diese Tabelle zugreifen und starten sie eine nach der anderen neu.

6.4.85 110: Locking-Fehler

Ursache

Der Rechner unterstützt das Sperren von Dateien nicht.

6.4.86 111: Blob nicht zu öffnen

Ursache

Beim Öffnen einer Tabelle wurde die zugehörige Blob-Datei nicht gefunden oder konnte (zum Beispiel wegen fehlender Rechte auf Datei-Ebene) nicht geöffnet werden.

6.4.87 112: Memo defekt

Ursache

Die zu einer Tabelle gehörende Memodatei kann zwar geöffnet werden, ist aber in ihrer internen Struktur gestört, so dass die Datenkonsistenz nicht gewährleistet ist.

Maßnahme

Nach dieser Fehlermeldung können Sie in der betroffenen Tabelle keine neuen Memos eingeben oder bestehende löschen. Sie können aber durch eine Umstrukturierung der Tabelle die Konsistenz wieder herstellen. Dabei gehen evtl. Daten verloren aber der noch brauchbare Teil der Informationen wird gerettet.

6.4.88 113: Blob defekt

Ursache

Die zu einer Tabelle gehörende Blob-Datei kann zwar geöffnet werden, ist aber in ihrer internen Struktur gestört, so dass die Datenkonsistenz nicht gewährleistet ist.

Maßnahme

Nach dieser Fehlermeldung können Sie in der betroffenen Tabelle keine neuen Bilder/Klänge eingeben oder bestehende löschen. Sie können aber durch eine Umstrukturierung der Tabelle die Konsistenz wieder herstellen. Dabei gehen evtl. Daten verloren aber der noch brauchbare Teil der Informationen wird gerettet.

6.4.89 114: Operation abgebrochen

Bedeutung

Der Anwender hat die Operation abgebrochen (z.B. mit Strg+U)

6.4.90 115: Kein Schreibrecht auf Datei

Bedeutung

Für die Datei, in die geschrieben werden soll, fehlt dem Anwender das entsprechende Recht auf Betriebssystem-Ebene.

6.4.91 116: Index noch in Gebrauch

Bedeutung

Der Index kann nicht gelöscht werden, weil er noch benutzt wird.

6.4.92 117: Ungültiges Schema für Tabelle

Ursache

Es wurde versucht eine Tabelle neu zu erzeugen oder zu restrukturieren, aber die neue Struktur der Tabelle war ungültig.

Maßnahme

Überprüfen Sie die Tabellenstruktur und versuchen Sie es erneut.

6.4.93 118: Fataler Fehler beim Aufruf der externen Prozedur

Ursache

Dieser Fehler tritt auf, wenn Sie eine Funktion aus einer externen Bibliothek aufrufen (*dllproc*). Der Aufruf führte in der Bibliothek zu einer schweren Ausnahmen. Dafür gibt es drei mögliche Gründe:

- Die *dllproc*-Anweisung stimmt nicht präzise mit der externen Funktion überein

- Beim Aufruf der externen Funktion wurden ungültige Parameter-Werte angegeben
- Die externe Funktion enthält einen Fehler.
- Maßnahme

Wenn dieser Fehler nach der Umstellung von einer früheren Version auf TurboDB Studio auftritt, könnte es an der Umsetzung der Zeichenketten-Parameter liegen. Lesen Sie im Kapitel über Upgrade den entsprechenden Abschnitt.

6.4.94 119: Externe Prozedur nicht gefunden

Ursache

Die mit *dllproc* deklarierte externe Prozedur konnte in der angegebenen Bibliothek nicht gefunden werden. Dafür gibt es drei mögliche Ursachen:

- Der Name der externen Prozedur wurde in der dllproc-Anweisung nicht korrekt angegeben
- Der Name der dynamischen Linkbibliothek wurde in der dllproc-Anweisung nicht korrekt angegeben
- Die vorliegende Version der Bibliothek enthält die Prozedur tatsächlich nicht (alte Version?)

6.4.95 122: Bei der SQL-Abfrage ist ein Fehler aufgetreten

Ursache

Der SQL-Befehl ist nicht korrekt.

Maßnahme

Prüfen Sie den Befehl auf seine Korrektheit.

6.4.96 123: Ausnahme in Turbo Datenbank

Ursache

Ein interner Fehler in Turbo Datenbank ist aufgetreten.

Maßnahme

Starten Sie *TurboDB Studio* neu und versuchen Sie es noch einmal. Wenn der Fehler wieder auftritt, wenden Sie sich an die Hotline. Für eine möglichst ausführliche Beschreibung des Fehler sind wir Ihnen sehr dankbar. Prüfen Sie auch, ob nicht schon eine Korrektur für diesen Fehler auf der dataWeb Web Site im Internet bereit liegt.

6.4.97 124: Veraltete Version der Tabellen-Datei

Ursache

Sie versuchen eine Tabelle zu öffnen, deren Tabellen-Dateiformat nicht mehr unterstützt wird.

Maßnahme

Benutzen Sie Visual Data Publisher oder TurboDB Viewer 4, um die Tabelle zu reparieren oder besser, sie nach Level 3 zu konvertieren.

6.4.98 125: Einer Konstanten kann nichts zugewiesen werden

Ursache

Die Anwendung hat versucht, einer konstanten Variablen einen neuen Wert zuzuweisen. Dies ist nicht erlaubt.

6.4.99 126: Links von . muß ein Objekt stehen

Ursache

Mit dem Punkt können Sie spezielle Prozeduren zu einem Objekt, z.B. einem Datenfenster aufrufen. Dazu muss es sich bei dem Ausdruck links von dem Punkt aber auch wirklich um ein Objekt handeln.

6.4.100 127: Array hat zuviele Elemente

Ursache

Die Gesamtgröße eines Arrays darf 64KB nicht überschreiben. Je nach Größe eines Eintrags ergibt sich dadurch eine unterschiedliche maximale Anzahl von Elementen. Bei Zahlen sind dies z.B. 8192. Die betreffende Array-Definition hat diese Obergrenze überschritten.

6.4.101 128: Die Volltext-Suchbedingung enthält einen Fehler

Ursache

Die Volltext-Suchbedingung ist fehlerhaft.

Maßnahme

Prüfen Sie auf Schreibfehler, und ob Sie bei Feldnamen, selbstdefinierten Prozeduren und Variablen die Groß-/Kleinschreibung beachtet haben.

6.4.102 129: Die Version der dBase-Datei ist unbekannt

Ursache

Es wurde versucht eine dBase-Datei zu lesen, aber die Dateiversion wurde von TurboDB nicht erkannt.

Maßnahme

Konvertieren Sie diese dBase-Tabelle mit einer geeigneten Anwendung in ein Format, welches von TurboDB erkannt wird.

6.4.103 130: Die Datei wird noch von einem anderen Anwender benutzt

Ursache

Eine Datei, die noch von einem anderen Anwender oder der selben Anwendung benutzt wird, kann nicht gelöscht oder umbenannt werden.

6.4.104 132: Unbekannte Klasse

Bedeutung

Die angegebenen Klasse ist nicht bekannt.

Ursache

Diese Fehlermeldung tritt meistens in der Prozedur CreateOleObject auf, wenn der Automatisierungs-Server auf dem Rechner nicht installiert ist oder der Name der Automatisierungs-Klasse falsch geschrieben.

6.4.105 133: Ungültige Objekt-Referenz

Ursache

Eine Objektvariable verweist auf ein Objekt, das nicht mehr existiert.

6.4.106 134: Dateikopf der Tabelle ist beschädigt

Ursache

Die Tabellendatei ist nicht mehr lesbar.

Maßnahme

Spielen Sie die Sicherung ein.

6.4.107 135: Fehler beim Aufruf einer Bibliotheks-Routine

Bedeutung

Beim Aufruf einer Oberflächenfunktion ist ein Fehler aufgetreten,

Ursache

Die Ursache ist im Fehlertext beschrieben.

6.4.108 136: Methode oder Eigenschaft nicht gefunden

Ursache

Das Progamm verweist auf eine Methode oder Eigenschaft eines Objektes, die nicht existiert.

6.4.109 137: Ungültiges Datum

Ursache

Sie haben eine Zeichenkette für ein Datum angegeben, dass den Regeln für eine Datumsangabe nicht enspricht.

6.4.110 138: Sprachtreiber wurde nicht gefunden, ist eine alte Version oder entspricht nicht der Spezifikation

Bedeutung

Die Tabelle benötigt einen Sprachtreiber, der aber nicht geladen werden kann.

Ursache

- Die Datei des Sprachtreibers ist nicht vorhanden.
- Der Sprachtreiber passt nicht zur Version von TurboDB Studio.
- Der Sprachtreiber ist nicht korrekt implementiert.

6.4.111 139: Das Argument einer Aggregations-Funktion muss nummerisch sein

Ursache

Es wurde eine Aggregationsfunktion mit einem Argument aufgerufen, das nicht nummerisch ist, zum Beispiel mit einer Zeichenkette.

6.4.112 140: Ungültige Zeitangabe

Bedeutung

Der angegebene Ausdruck ist keine gültige Zeitangabe.

Ursache

Das Format für die Zeitangabe stimmt nicht. Beispielsweise wird ein anderes Trennzeichen als der Doppelpunkt verwendet.

6.4.113 141: Keine Berechtigung, Datei zu erstellen

Bedeutung

Die Datei kann nicht erstellt werden, weil das Recht dazu fehlt.

Ursache

Das Recht zum Erstellen der Datei fehlt auf Betriebssystem-Ebene.

6.4.114 142: Keine Berechtigung, Datei zu lesen

Bedeutung

Die Datei kann nicht geöffnet werden, weil das Recht dazu fehlt.

Ursache

Das Leserecht für diese Datei fehlt auf Betriebssystem-Ebene.

6.4.115 143: Feld hat die Größe null

Ursache

Beim Erzeugen oder Restrukturieren einer Tabellen haben Sie ein Feld mit der Speichergröße 0 definiert. Das macht natürlich keinen Sinn.

6.4.116 144: Unbekannter Feldtyp

Bedeutung

Der angegebene Datentyp für eine Tabellenspalte ist ungültig.

6.4.117 145: Auto-Nummer-Feld hat keine Anzeige

Bedeutung

Die Tabelle kann nicht erzeugt oder umstrukturiert werden, weil die Definition ein Auto-Nummer-Feld ohne Ausdruck für den id-Index enthält.

Ursache

Der Ausdruck für den id-Index wurde nicht angegeben.

6.4.118 146: Ungültiger Verbund

Bedeutung

Der statische Link in einer Relations-Anweisung ist ungültig.

Mögliche Ursachen

- Die Syntax entspricht nicht den Regeln.
- Es sind ungültige Tabelle- oder Spaltennamen enthalten.
- Die Typen der Indexausdrücke passen nicht zusammen.

6.4.119 147: Alle Tabellen einer Sitzung müssen den selben Sprachtreiber benutzen

Ursache

Alle Tabellen, auf die in einer Sitzung zugegriffen wird, müssen die selbe Sprache haben oder Sprach-neutral sein. Es wurde versucht, eine Tabelle zu öffnen, die eine andere Sprache hat, als die schon geöffneten Tabellen dieser Sitzung.

Maßnahme

Öffnen Sie die Tabelle in einer anderen Sitzung oder ändern sie ihre Sprache durch umstrukturieren.

6.4.120 149: Keine Berechtigung, einen Index zu erstellen

Ursache

Sie wollen einen neuen Index erstellen, aber in der Tabelle fehlt die entsprechende Berechtigung.

6.4.121 150: Zu viele Werte in der Liste

Ursache

In einem SQL *insert*-Befehle übersteigt die Anzahl der Werte in der *values*-Liste, die Anzahl der Spalten der Tabelle bzw. der Einträge in die Spaltenliste.

6.4.122 151: Equate join hat genau eine Tabelle auf jeder Seite des =

Ursache

Ein statischer Link, zum Beispiel in einem Relationskommando, muss sich auf jeder Seite des Gleichheitszeichens auf genau eine Tabelle beziehen.

6.4.123 152: DBase III-Dateien können diese Datenstruktur nicht aufnehmen

Ursache

Die TurboDB-Tabelle kann nicht nach DBase exportiert werden, weil das DBase Dateiformat die Datenstruktur der TurboDB-Tabelle nicht abdecken kann.

6.4.124 153: Fehler in externer Tabelle

Ursache

Eine externe Tabelle ist eine Tabelle, in die TurboDB exportiert, oder aus der es importiert. Bei einem solchen Lese- oder Schreibvorgang ist ein Fehler aufgetreten. Meistens hat dieser mit dem Dateisystem, mangelnden Recht, nicht ausreichend Speicherplatz o.ä. zu tun.

6.4.125 154: Relations-Tabelle konnte nicht geöffnet werden

Ursache

Es wurde eine Tabelle mit Relationsfeld geöffnet. Die zugehörige Relations-Tabelle konnte allerdings entweder nicht gefunden oder nicht geöffnet werden.

6.4.126 155: Eine oder mehrere Dateien konnten nicht gelöscht werden

Ursache

Bei einer Operation mit mehreren Dateien (zum Beispiel Tabelle umbennen, löschen, umtrukturieren) konnen einige der Dateien nicht gelöscht werden.

Maßnahme

Die ist kein kritischer Fehler, allerdings sollten Sie diese Datein nachträglich per Hand entfernen.

6.4.127 156: Memo/Blob-Datei überschreitet maximale Größe

Ursache

Memo- und Blob-Dateien haben eine maximale Größe, die sich aus der Größe eines einzelnen Blocks errechnet.

Maßnahme

Wenn diese Größe erreicht ist, müssen Sie die Blockgröße erhöhen, bevor Sie wieder Daten eingeben können.

6.4.128 157: AutoInc-Feld darf nicht geändert werden

Ursache

Der Wert eines *AutoInc*-Feldes darf nicht durch Zuweisung geändert werden. Er wird automatisch berechnet.

6.4.129 158: Ungültiger Zeitstempel

Ursache

Es wurde eine Zeichenkette für einen Zeitstempel angegeben, die dem Format nicht entspricht.

Maßnahme

Geben Sie den Zeitstempel in einem korrekten Format, z.B. als *24.8.2001_13:45:20* an oder fangen Sie diesen Laufzeitfehler mit einem *try*-Block ab.

6.4.130 159: Sprachtreiber wird vom Betriebssystem nicht unterstützt

Ursache

Für die Tabelle wurde eine Sprache angegeben, die auf dem aktuellen Betriebssystem nicht vorhanden ist.

6.4.131 160: Schreibrecht für Tabelle fehlt

Ursache

Die Anwendung hat kein Schreibrecht für die Tabelle. Dieses wurde entweder beim Öffnen nicht angefordert, oder die Tabelle ist auf der Festplatte schreibgeschützt.

6.4.132 161: Die Auswertung des Ausdrucks liefert keinen Wert

Ursache

Es wurde ein Ausdruck ausgerechnet, von dem ein Ergebnis erwartet wird, zum Beispiel, um es einer Variable zuzuweisen. Der Ausdruck liefert aber kein Ergebnis, beispielsweise weil es sich um eine Prozedur ohne Rückgabewert handelt.

6.4.133 162: Das Datenfeld hat nicht den erforderlichen Typ

Ursache

An dieser Stelle wird ein Datenfeld vom angegebenen Typ erwartet. Dies ist aber nicht erfüllt.

6.4.134 164: Unbekannter Typ für Variablen und Parameter

Ursache

Der angegebene Datentyp ist nicht bekannt.

6.4.135 165: Fehler bei der Ausführung einer OLE-Funktion

Ursache

Es wurde von TurboPL aus ein OLE-Automatisierungs-Aufruf gestartet, der mit einem Fehler zurückgekehrt ist.

6.4.136 166: Modul benutzt sich indirekt selbst

Ursache

Das TurboPL-Modul benutzt mit *uses* ein anderes Modul, welches dann wieder das anfängliche TurboPL-Modul benutzt. Es kann auch sein, dass sich dieser Kreis über mehr als zwei Module erstreckt.

Maßnahme

Sie müssen die Prozeduren so auf die Module verteilen, dass Sie eines dieser *uses*-Kommandos entfernen können.

6.4.137 167: Datensatz ist ungültig

Ursache

In der Tabelle ist eine Gültigkeitsbedingung definiert und es wird versucht einen Datensatz einzutragen, der diese nicht erfüllt. Die Gültigkeitsbedingungen können Sie mit TurboDB Viewer oder TurboDB Studio Datenbankfenster ansehen.

6.4.138 168: Blob-Feld erwartet

Ursache

Der Prozeduraufruf erwartet als Argument ein Blob-Feld. Stattdessen wurde etwas anderes angegeben.

6.4.139 169: Memo-Feld erwartet

Ursache

Der Prozeduraufruf erwartet als Argument ein Memo-Feld. Stattdessen wurde etwas anderes angegeben.

6.4.140 170: Mindestens eine Tabelle ist gesperrt

Ursache

Die Anwendung will eine Sperre auf eine Tabelle setzen, die schon gesperrt ist.

6.4.141 171: Die maximale Kapazität ist erreicht

Ursache

Eine Tabelle (d.h. eigentlich deren Indexe) haben eine maximale Kapazität, die beim Anlegen des Index festgelegt wird.

Maßnahme

Wenn der Index voll ist, müssen Sie die Tabelle bzw. den Index restrukturieren, um die Kapazität zu erhöhen.

6.4.142 172: Der nicht aggregierte Ausdruck muss in der GROUP BY-Liste enthalten sein

Ursache

Wenn Sie in einem SQL *select*-Kommando ein *group by* verwenden, muss jede der hinter *select* angeführten Spalten entweder in der *group by*-Klausel enthalten sein oder eine Aggregation (*max*, *min*, *avg*, *count* usw.) darstellen.

6.4.143 173: Ungültiger Spaltenname

Ursache

Maximale Länge ist 40 und Spaltennamen für Level 3 und vorher dürfen weder Leerzeichen noch andere Sonderzeichen enthalten

6.4.144 174: Der Bezeichner ist nicht eindeutig

Bedeutung

TurboPL kann nicht sicher feststellen, zu welchem Objekt der Bezeichner gehört.

Ursache

Sie verwenden den selben Namen für zwei verschiedene Objekte, zum Beispiel für einen Feldnamen und eine Variable.

Maßnahme

Bitte stellen Sie Feldnamen den Tabellennamen voran und fügen Sie ggf. ein Dollar-Zeichen ($) hinzu, um zwischen Feldern und Variablen oder Prozeduren zu unterscheiden

6.4.145 175: Auf die Tabelle "<Name>" wird von dieser Sitzung noch zugegriffen

Bedeutung

Die Anwendung versucht, eine Tabelle zu schließen, auf welche die selbe Anwendung noch zugreift.

Ursache

Mögliche Gründe dafür sind:

- In TurboDB Studio-Anwendungen: Es ist noch ein Formular für diese Tabelle geöffnet.
- In Delphi-Anwendungen: Es ist noch eine *TTdbTable*- oder *TTdbQuery*-Komponente mit dieser Tabelle aktiv.
- In Visual Studio-Anwendungen: Es gibt noch einen *DataReader* oder ein *TurboDBRecordSet* für diese Tabelle, bei dem Close nicht aufgerufen wurde.

6.4.146 176: Ungültiger Index-Name

Ursache

Index-Namen müssen gültige TurboDB Bezeichner sein und dürfen nicht mit "sys_" oder "tmp_" beginnen"

6.4.147 177: Nicht genügend Argumente für Prozeduraufruf angegeben

Ursache

Die aufgerufene Prozedur benötigt mehr Argumente als angegeben wurden.

6.4.148 178: Division durch 0

Ursache

Es wurde versucht durch 0 zu teilen, dies ist nicht erlaubt.

6.4.149 179: Für die Integritätsregel wurde kein Datensatz in der Vater-Tabelle gefunden

Ursache

Für die Tabelle ist eine Regel zur referentiellen Integrität definiert, die eine Beziehung zu einer anderen Tabelle herstellt. Die gewünschte Aktion würde diesen Bezug zerstören, weil kein passender Datensatz mehr in der anderen Tabelle vorhanden wäre. Deshalb wird sie nicht durchgeführt.

6.4.150 180: Die Operation verletzt eine Integritätsregel

Ursache

Für die Tabelle sind Integritätsregeln definiert, die Sie beispielsweise im TurboDB Viewer oder im TurboDB Studio Datenbankfenster einsehen können. Die gewünschte Aktion würde zur Verletzung einer solchen Regel führen und kann deshalb nicht ausgeführt werden.

6.4.151 181: Der Datensatz ist nicht im Editiermodus (rufen Sie NewRec oder ModifyRec auf)

Ursache

Wenn Sie den Wert von Feldern ändern wollen, muss der Datensatz im Editiermodus sein. Dies ist nicht der Fall.

6.4.152 182: Der Kursor steht nicht auf einem gültigen Datensatz

Ursache

Die Position des Tabellenkursors ist außerhalb der Tabelle. In diesem Zustand können Sie den gewünschten Befehl nicht ausführen.

6.4.153 183: Es läuft schon eine Transaktion

Ursache

Sie können keine Transaktion starten, wenn schon eine gestartet ist.

Maßnahme

Sie müssen zuerst die laufende Transaktion beenden.

6.4.154 184: Die Eltern-Tabelle "<Tabelle>" konnte weder gefunden noch geöffnet werden

Ursache

Beim Öffnen einer Tabelle mit referentieller Integritätsbeziehung konnte die Tabelle, auf die sich die Integrität bezieht nicht gefunden oder geöffnet werden.

Maßnahme

Stellen Sie entweder den Zugriff auf die Eltern-Tabelle wieder her oder entfernen Sie die Integritätsbeziehung der Tabelle im TurboDB Viewer.

6.4.155 185: Die Sperren-Datei "<Name>" ist entweder ungültig oder nicht kompatibel mit dieser Anwendung. Versuchen Sie, sie zu löschen.

Bedeutung

Beim Öffnen einer Tabelle im Netzwerk-Modus hat TurboDB festgestellt, dass schon eine Net-Datei vorhanden ist, welche aber nicht dem aktuellen Format für Net-Dateien entspricht.

Ursache

Entweder ist die Tabelle schon durch Anwendungen, die mit einer früheren (oder späteren) Version von TurboDB laufen geöffnet oder die Tabelle ist noch nicht geöffnet und die Net-Datei wurde aufgrund eines Absturzes von der vorherigen Anwendung nicht gelöscht. Insbesondere kann TurboDB Studio 4 die net-Dateien von Visual Data Publisher nicht lesen, das es neue Merkmale unterstützt.

Maßnahme

Im ersten Fall müssen Sie dafür sorgen, dass die Anwendung nur von Anwendungen derselben TurboDB-Version geöffnet wird. Im anderen Fall genügt es, die net-Datei und die mov-Datei (bzw. rnt und rmv) manuell zu löschen.

6.4.156 186: Das Passwort ist ungültig (zu kurz, zu lang oder der nummerische Schlüssel fehlt bei klassischer Verschlüsselung)

Bedeutung

Beim Erzeugen oder Umstrukturieren einer Tabelle wurde ein ungültiges Passwort für die Verschlüsselung angegeben.

Mögliche Ursachen

- Das Passwort hat zu wenig Zeichen.
- Das Passwort hat zu viele Zeichen.
- Das Passwort für die klassische Verschlüsselung enthält keinen durch Semikolon abgetrennten nummerischen Code.

Index

- 2 -

- 3 -

- B -

- C -

- E -

- F -

- G -

- H -

- I -

- M -

- N -

- O -

- T -

- U -

- X -

- Y -

- Z -